Aria: Autonomous Real-Time Interactive Architecture

By
Cameron Blanchard - 100886476
Jeremy Dunsmore - 100889935
Peter Mark - 100886038
Matthew Maynes - 100882216

Supervisor: Babak Esfandiari

A report submitted in partial fulfillment of the requirements
of SYSC-4907 Engineering Project

Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University

February 1, 2017

Abstract

Acknowledgements

Table of Contents

List of Figures 5
List of Tables 5
1 Project Background 6
1.1 Introduction 6
1.2 Problem Motivation 6
1.3 Problem Statement 6
1.4 Proposed Solution 6
1.5 Accomplishments 6
1.6 Overview of Report 6
2 Engineering Project 7
2.1 Health and Safety 7
2.2 Engineering Professionalism 7
2.3 Development Process 7
2.3.1 Introduction 7
2.3.2 Source Code Management 7
2.3.3 Branching and Code Reviews 8
2.3.4 Continuous Integration 8
2.3.5 Issue Tracking 8
2.3.6 Testing Practices 8
2.4 Individual Contributions 9
2.4.1 Project Contributions 9
2.4.2 Report Contributions 9
2.4.3 Appendix Contributions 9
3 Technical Sections 11
3.1 Requirements 11
3.1.1 Overview 11
3.1.2 Overall Description 11
3.1.3 System Features 12
3.1.4 External Interface Requirements 13
3.1.5 Other Nonfunctional Requirements 15
3.2 Scenarios 16
3.2.1 Music Automation 16
3.2.2 Efficient Lights and Temperature 18
3.2.3 Coffee Automation 20
3.2.4 Effortless Home Lighting 22
3.3 Design 26
3.3.1 Introduction 26
3.3.2 System Components 26
3.3.3 Deployment 28
3.3.4 Database 28
3.3.5 Discovery 30
3.3.6 Requests 31
3.3.7 Remote User Interface 33
3.3.8 IPC Protocol 34
3.3.9 Database Interface 43
3.3.10 Gateway REST API 45
3.3.11 Gateway Websocket Protocol 51
3.4 Development 53
3.4.1 Deployment 53
3.4.2 Technologies 54
3.4.3 Hub Implementation 55
3.4.4 Communication with Z-Wave Devices 58
3.4.5 Gateway Implementation 59
3.4.6 Remote Implementation 59
3.4.7 Lighting Scenario Data 61
3.5 Testing 63
3.5.1 Hub Testing 63
3.5.2 Remote Testing 63
3.6 Project Progress 64
3.6.1 Proposed Timeline 64

3.6.2 Project Status 65
3.7 Data Flows 66
3.7.1 Add Device Data Flow 66
4 Conclusion and Recommendations 69
5 Appendix 70
A Automation Systems 70
Background 70
A-1 Insteon 70
A-2 Wink 71
A-3 SmartThings 73
A-4 Apple HomeKit 74
A-5 Summary 74
B Computing Devices 74
Background 75
Relation to System 75
B-1 Arduino Uno 75
B-2 Arduino 101 75
B-3 Arduino Pro 76
B-4 Arduino Micro 76
B-5 Comparison of Arduinos 76
B-6 Raspberry Pi Zero 77
B-7 Raspberry Pi 1 Model A+ 78
B-8 Raspberry Pi 2 Model B 78
B-9 Raspberry Pi 3 Model B 78
B-10 Comparison of Raspberry Pi 79
B-11 BeagleBone 80
B-12 BeagleBone Black 80
B-13 BeagleBone Green 80
B-14 Comparison of BeagleBone 81
B-15 Summary of Devices 81
C Communication Protocols 82
Background 82
C-1 ZigBee 82
C-2 Z-Wave 83
C-3 Insteon 83
C-4 WiFi 84
C-5 Bluetooth 84
C-6 Summary 84
D Custom-Built Devices 85
Background 85
Devices of Interest 86
D-1 Motion Sensor 87
D-2 Variable-Voltage Switch 88
D-3 Light Sensor 89
D-4 Thermostat 90
D-5 Alarm Clock 90
D-6 Coffee Makers 90
D-7 Summary 91
E Smart Devices 91
Background 91
Relation to System 91
E-1 WeMo 92
E-2 Nest Thermostat 93
E-3 Honeywell VisionPro Thermostat 93
E-4 Wireless Speakers 93
E-5 Smart TV 94
E-6 Philips Hue 94
E-7 Osram LIGHTIFY 95
E-8 Aeotec Light Bulbs 96
E-9 Aeon Labs 97
E-10 Passive Infrared Sensor 98
E-11 Spruce Irrigation 98
E-12 OSO PlantLink 98
E-13 Summary of Evaluation 99

F Purchases List 99
G Raw Data 100
G-1 Z-Wave Device Data 100
6 References 104

List of Figures

Figure	Author(s)	Editor(s)
Code Lifecycle	Cameron	
System Use Case	Cameron	Matthew
Remote Use Case	Matthew	Cameron
Light and Temperature Scenario Sequence	Matthew	
Coffee Scenario Sequence	Jeremy	

List of Tables

Table	Author(s)	Editor(s)

1 Project Background

1.1 Introduction

The field of home automation systems is a young and expanding market. Home automation, or a "Smart Home" is a building that utilizes physical devices to automate tasks and collect data. The devices used in the home are connected through a network medium and act as an Internet of Things (IoT) [1]. As technology evolves, embedded devices used in smart homes are becoming more sophisticated, offering more features and better information capturing. However, these advancements come with a new set of challenges. Higher costs of ownership and more technical experience is required to operate these new devices keeping the technology from being adopted by the general public [2].

1.2 Problem Motivation

This project aims to simplify the smart home for an average consumer. Through the use of automated configuration, flexible device requirements and automated programming, the concept of the smart home could be accessible to a non-technical user.

1.3 Problem Statement

Smart home automation is not accessible to non-technical users. Setting up IoT devices can require knowledge of many levels of the device's inner-workings and how it communicates. Additionally, scheduling behaviours of devices for the smart home requires operation of a technical user interface to control, schedule and monitor their behaviour.

1.4 Proposed Solution

Aria (Autonomous Real-Time Interactive Architecture) is a project that is modeled upon existing home automation systems with the focus of user interaction, device configuration and device automation through the use of machine learning. By removing the technical requirements from the user, we expand the market for home automation consumers. Beyond the essentials of monitoring and controlling a home, the Aria system aims to provide an element of automation based on contextual learning from the user's interactions with the system.

1.5 Accomplishments

1.6 Overview of Report

This report outlines the details of the Aria project, illustrating the design and development of the system. It provides the technical background and details about the decisions made in its construction. The report also outlines the engineering practices used to work effectively in a team to complete this project.

2 Engineering Project

2.1 Health and Safety

2.2 Engineering Professionalism

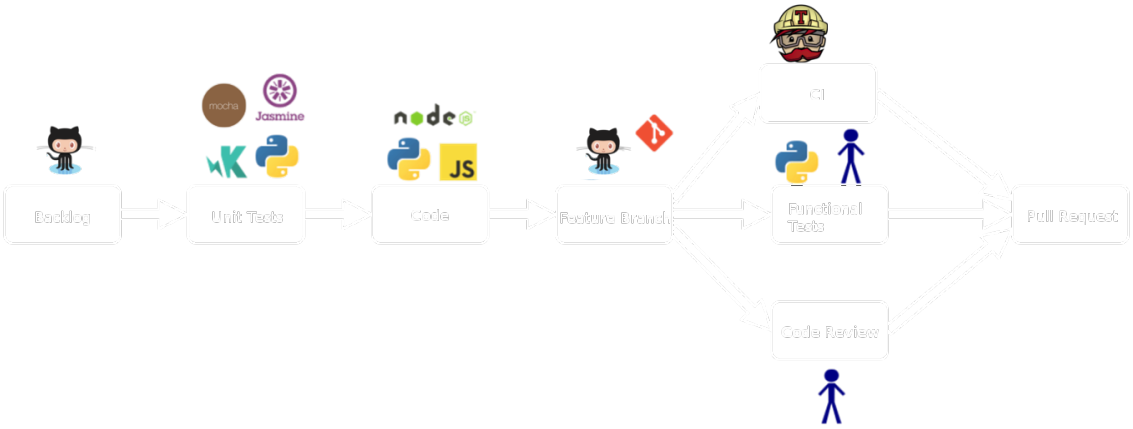
2.3 Development Process

2.3.1 Introduction

An important part of any sizeable software project is the selection of an appropriate development process. A development process imposes a particular structure on the development of a product. The process used for the development of this project was created with the following factors in mind:

- Project lifespan: Work on the project will last at least 8 months; other developers may also continue working on the project after the two school terms are complete
- Team Size: There are four team members working on the project, the process needs to keep everyone synchronized and organize the assignment of work to team members
- Reporting: The final project report is critical to the project's success, the process should make the report easier to write in a consistent manner
- Code Quality: The process should ensure that work products are of the quality expected of a fourth year project.
- Maintainability and Extensibility: The end result of the project should be maintainable and extensible; a developer who is new to the project should have an easy time developing for it.
- Ease of Implementation: The process should not be an obstacle to development

Given these goals, the process used for this project incorporates some elements of Extreme Programming and other Agile methodologies. Use of practices which are widely accepted in industry also ensures that new developers are comfortable working on the project. The process prescribes some practices to be used for: Source Code Management, Code Reviews, Issue Tracking, and Testing.



2.3.2 Source Code Management

The project uses a source code management tool to achieve the following goals:

- Allows many team members to work on the same codebase and maintain a consistent master copy of the project
- Track changes to files in order to locate changes that may have introduced bugs
- Allows team members to work independently and simultaneously

This project uses git because it provides solutions to all of these issues. Git is familiar to all team members, in addition to being a de-facto standard in the open source software community. Using git over other SCM tools makes the process easier to follow, and improves the experience of any new developer that might choose to maintain the project.

The project's git repository is hosted on Github. Github allows the team to access the project anywhere, and makes sharing code with new developers simple. Additionally, Github provides some project management tools that can be used elsewhere in the process, as explained in subsequent sections.

2.3.3 Branching and Code Reviews

We make strategic use of a git feature, branching, in order to ensure that code which makes it into the official version of our software is of the quality expected of a fourth year project. A technique we are using to ensure code quality is peer code review. By frequently reviewing code, we can share knowledge between team members with different amounts of experience in particular technologies.

In order for code reviews to be effective, we believe that the review should be as short as possible, and focused on a particular feature. This ensures that reviewers are motivated to provide high quality feedback. In order to encourage this, we have adopted the "feature branches" branching strategy. Work on a particular feature is submitted to its own branch; branches are merged to master once work on the feature has been completed. A code review is required before any branch may be merged into master. Feature branches helps to ensure that code reviews are short and focused.

2.3.4 Continuous Integration

A practice which complements the branching strategy adopted by this project is continuous integration. In combination with test automation, continuous integration allows team members to get early feedback about changes they make to the codebase. Proper use of continuous integration can ensure that the project's master copy remains in a working state at all times. One of the benefits of maintaining working software at all times is that the project is always demo-able.

The continuous integration tool chosen for this project is Travis CI. Travis CI is a free hosted CI system. Travis can be easily integrated with Github repositories, and provides quick and reliable feedback about the build state of the system.

Github rules were created to ensure that any code which is merged into the master branch does not break existing work. Merges to master are blocked until Travis has successfully run unit tests on the merged code.

2.3.5 Issue Tracking

In order for the team to measure the progress of the project, and to ensure that tasks are not forgotten, it is important for this process to incorporate a process for tracking tasks. Issue tracking may also simplify the task of writing the final report by providing a list of the tasks completed during the project.

Another advantage of issue tracking is that team members can be explicitly assigned to specific issues to work on; this ensures that efforts are not duplicated.

A good issue tracking tool:

- Provides a visible list of things to do
- Allows tasks to be ordered so that team members know what to do next
- Provides a forum for discussion of tasks

Github provides an issue tracking system which can be used to accomplish these goals. The Github issue tracking system is not ideal for some tasks, such as prioritization of issues. In the interest of minimizing the number of tools required to follow this process, however, Github issue tracking was selected as the team's issue tracking tool.

Github issues are organized into groups of tasks which must be completed during each one-week iteration of development. Issues are prioritized using labels (minor, major, critical, blocker). Issues are assigned to a team member when work begins; when work on an issue is completed the issue is closed using a commit message.

2.3.6 Testing Practices

Agile processes generally encourage a short feedback loop for developers (problems with code should be exposed as early as possible). This project uses principles of test-driven development (TDD) and behaviour-driven development (BDD) in order to ensure that development remains focused on problems which are relevant to the end user. Any change to the code should be accompanied by an automated test or set of tests which exercises **end user functionality**. That is, automated tests should function as an executable specification of requirements for the unit under test.

2.4 Individual Contributions

2.4.1 Project Contributions

2.4.2 Report Contributions

Section	Author(s)	Editor(s)	Updated
Development Process	Cameron	Peter	November 6, 2016
Technical Requirements	Cameron & Matthew	Peter	October 10, 2016
Scenario Introduction	Cameron	Matthew	October 10, 2016
Music Scenario	Cameron & Matthew	Peter	November 6, 2016
Temperature Scenario	Matthew	Peter	November 6, 2016
Coffee Scenario	Jeremy & Matthew	Peter	November 5, 2016
Lighting Scenario	Peter	Matthew	November 28, 2016
Design Introduction	Matthew		October 30, 2016
Component Design	Matthew	Jeremy & Cameron	November 27, 2016
Deployment Design	Matthew		October 30, 2016
Database Design	Peter		November 27, 2016
Device Discovery	Matthew		October 30, 2016
Device Requests	Peter	Matthew	October 30, 2016
Remote User Interface	Matthew		November 13, 2016
IPC Protocol	Matthew & Cameron	Jeremy	November 27, 2016
Database Interface	Peter	Matthew	November 27, 2016
REST API Design	Matthew		November 27, 2016
Websocket Design	Matthew		November 27, 2016
Deployment	Matthew	Cameron & Peter	November 6, 2016
Technology	Matthew	Peter	November 6, 2016
Hub Implementation	Jeremy		November 27, 2016
Gateway Implementation	Matthew		November 27, 2016
Remote Implementation	Matthew		November 28, 2016
Testing	Cameron		November 27, 2016
Hub Testing	Cameron	Peter	November 27, 2016
Remote Testing	Matthew		November 28, 2016
ZWave Communication	Cameron	Matthew	December 6, 2016
Lighting Scenario Data	Cameron & Jeremy		February 1, 2017
Add Device Workflow	Peter & Jeremy	Matthew	January 30, 2017

2.4.3 Appendix Contributions

Section	Author(s)	Editor(s)	Updated
A Automation Systems	Jeremy & Matthew	Peter	January 31, 2017
B Computing Devices	Matthew	Cameron	February 1, 2017
C Communication Protocols	Peter	Cameron & Matthew	October 30, 2016
D Custom Devices	Cameron	Peter & Matthew	October 30, 2016
E Smart Devices	All		October 29, 2016
F Shopping List	Jeremy		November 13, 2016
G Raw Data	Jeremy		January 31, 2017

3 Technical Sections

3.1 Requirements

3.1.1 Overview

Introduction

The Autonomous Real-Time Interactive Architecture (ARIA) will allow a homeowner to set up a collection of devices in their home which will automatically control their environment and automate common tasks. Task automation in the system will not require any user configuration, instead tasks will be automated based on the user's interaction with devices in the system.

The system will consist of a hub device with a simple interface that a homeowner can connect to their home network. After connecting the hub, the homeowner can add enabled devices for the system to control by simply connecting them to the network. As new devices are connected, their input will be used to make more predictive decisions about user behaviour.

Project Scope

The purpose of the system is to make home automation as easy as possible to set up. Many existing home automation systems require some form of programming from the user, in the form of a schedule or explicit scenarios which describe how the devices connected to them should behave. This project will improve upon such systems by inferring the correct state of connected devices from data collected during the homeowner's routine use of the devices.

Activities which are "In Scope":

- Provide a range of devices that can be installed in a home which showcase the learning capabilities of the system.
- Create a hub device that collects data from installed sensors, and uses this data to infer the desired values of actuators.
- Provide a simple interface which allows the homeowner to toggle on/off automated control of devices.
- Provide a user interface for controlling the smart devices in the home.
- Provide a user interface for monitoring the state of the home and the devices in it.

3.1.2 Overall Description

Product Perspective

Systems are becoming more readily available in the general market place. This system builds upon traditional home automation systems by adding true automation in the form of learning. Unlike traditional systems, this system observes the homeowner's interactions with devices and automatically makes decisions based on historical behaviours. By combining the fields of machine learning and home automation, the our system will provide an end user with a simplified smart home experience.

Product Functions

To provide simplicity and seamless interaction, the system must be easy to use and highly interactive. The system will provide a learning hub that will be the base of computation and communication for all other components in the system.

The system will have many different smart devices. These devices can be sensory inputs or controls for a task. To allow for expansion of the automation system, it must be able to accept new devices. The system must then retain its previous model of the user's interactions but add in the new device as evidence for predicting future behaviours. For example, a system might include a light sensor, thermostat and curtain puller. If the ambient light outside was to drop and the user closed the curtains then the system might predict that a change in light corresponds to that action. If the user then changes the temperature, the system may relate light to temperature as well. If later a temperature sensor was added to the system and the user changes the temperature when it is too cold, the system will use this information to predict future actions.

To be able to monitor and control the system, the learning hub must be controllable using a graphical interface. This interface will be provided in the form of a web interface and allow the user to view the state of the system and manually control any connected device. This interface can be used for manually configuring desired behaviours as well as for enabling the learning mode of the system. The interface must allow the user to view the state of all devices in the system as well as view their recorded interactions.

User Classes

While the primary audience of the system is a homeowner, it is also for building owners, nursing home residence, or anyone

who needs building automation. The end user of this product is intended to be non-technical users who want simple control and automation of their building. This product will also provide utilities for more technically proficient users who wish to create their own devices that communicate to the system.

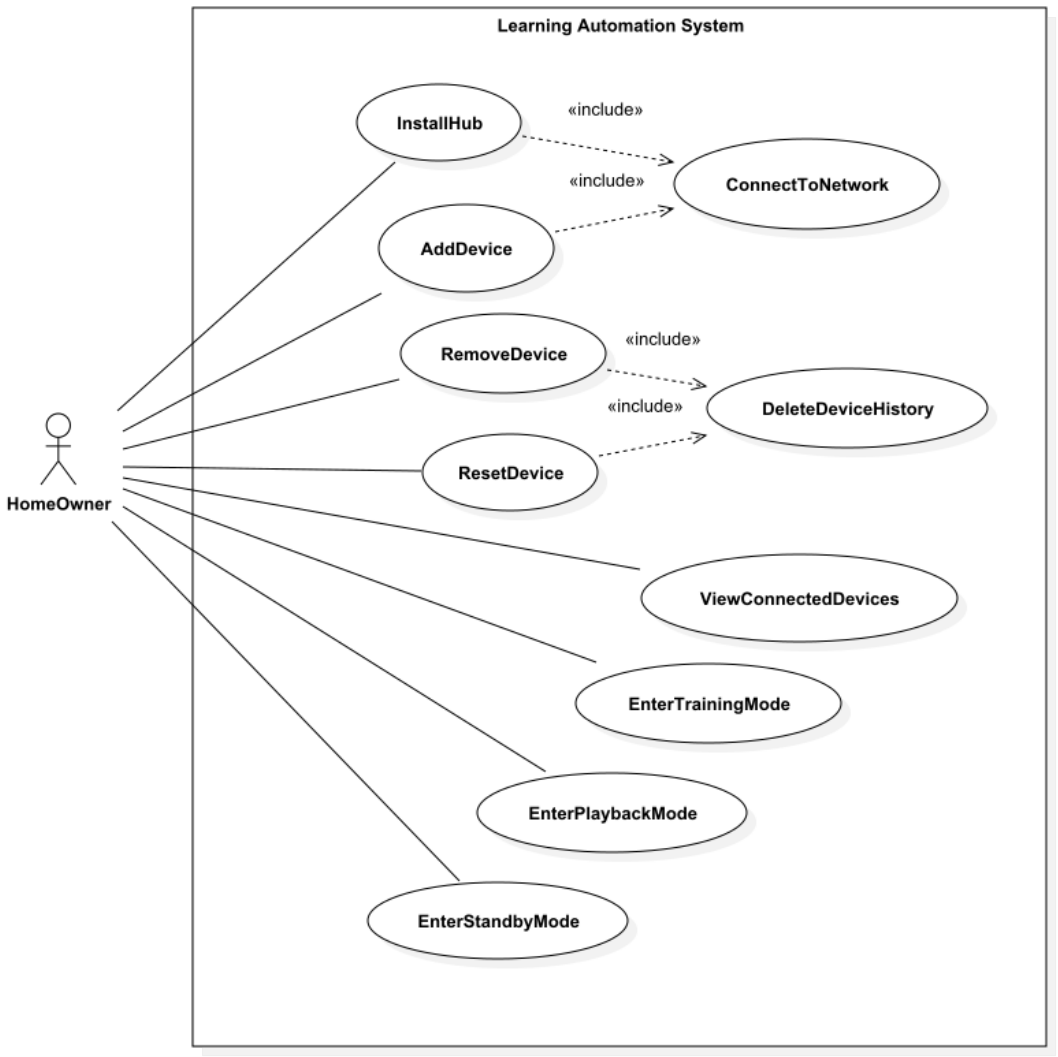
Base User (Non-Technical Users)

- End user of the system that needs simple interface to use system
- Will want low maintenance to keep system running and expect system to perform correctly

Developers (Technical Users)

- Will require technical documentation about the system
- Will require an API to communicate to custom devices for the system
- May also be a base user

3.1.3 System Features



Install Hub

The user installs the learning hub in their home in order to enable automation of their smart devices.

1. User plugs hub into outlet and turns power on
2. User connects hub to a home network using Ethernet
3. Hub provides confirmation that system is online

Add Device

Devices can be added to the system simply by powering them on and connecting to the network.

Precondition: A learning hub must be installed in the user's home.

1. User plugs in device and turns power on
2. Device discovers network
3. Hub discovers device and provides confirmation

Postcondition: The device's state will now be used as input in learning mode. If the device contains an actuator, the actuator will be controlled by the learning hub in normal mode.

Enter Learning Mode

The user enters learning mode in order to indicate to the system that it should begin recording changes in the state of connected devices, without attempting to control them. Learning mode accomplishes the user's goal of configuring the system without manual programming.

1. User selects enter learning mode
2. While the system is in learning mode, the system will record the user's interactions with connected devices.
3. When the user selects normal mode or standby mode, the system exits learning mode.

Postcondition: The system saves changes in the state of connected devices.

Enter Normal Mode

The user enters normal mode in order to instruct the system to begin controlling connected devices.

1. User selects enter normal mode
2. System exits the currently active mode
3. System begins controlling connected actuators, using the data collected during learning mode to infer the desired state of the system.

Postcondition: The system maintains control over connected actuators.

Enter Standby Mode

The user enters standby mode in order to instruct the system that control over connected devices should be halted, and changes in the state of devices should not be accepted as learning data. Standby mode allows a user to control their devices under exceptional circumstances without learning the system to perform an incorrect task.

1. User enters standby mode
2. System exits the active mode

Postcondition: System does not accept learning data, System does not modify the state of devices

Remove device

Devices will stop recording when removed from the smart learning network. To remove the history of the device, the user can delete it using the remote interface.

1. User disconnects device from the network
2. If the user wishes to remove the device permanently, include use case Reset Device

Reset Device

If the input of a device is causing unexpected or undesired output then it can be reset by the user through the remote interface.

1. User logs in to remote interface
2. User selects a device
3. User selects reset device
4. System erases the saved historical states of the device

Postcondition: States of the selected devices from before the reset are no longer used to infer states in normal mode.

3.1.4 External Interface Requirements

User Interfaces

Remote Interface

The remote interface will allow the user to control the system and any device that is connected in the network. The remote interface will be a web application that is served to the user's computer from the learning hub.

Accessing the Interface

To load the web application, the user will navigate through a web browser to the address of their learning hub. The hub will then provide the remote interface and prompt the user with a login. The first time the user opens the hub control page they will be prompted to create an account.

Viewing and Controlling Devices

The primary use of this remote interface will be to observe the state of the system as well as control any connected device. The remote interface must provide access to the recent history of all interactions that have occurred in the system. The interface must provide a mechanism for searching the logs and grouping them based on time and device.

The system must also be able to control the devices that are connected to it. The remote interface must provide the appropriate controls for each device that is connected to the system.

Controlling the Hub

The interface must graphically provide methods to customize the learning hub's operation. This will include properties related to the system such as network connections, login options and any other hub specific items.

Exiting the Remote

Once the user is finished with the remote interface, they can log out or simply close the web application. For security reasons, if a user is inactive in their session for more than a set amount of time then they will be logged out automatically.

Hardware Interfaces

Learning Hub Interface

To maximize simplicity, the learning hub interface will have a clean and minimal interface. It will provide the user with three control buttons and one reset button. The control buttons will allow the user power off and on the device as well as toggle the state between learning mode, normal mode and standby.

The hub will provide user feedback with a single multi-colour LED. The LED will be used to indicate the state of the device. There will be a state for all three operation modes; learning, normal and standby. If there is an error in the device, the LED can be used to indicate the error.

There will also be two external ports on the device. One will be used to power the device from a standard home wall outlet. The other can be a standard Ethernet port and be used to connect to the network.

3.1.5 Other Nonfunctional Requirements

Performance Requirements

Device Communication Range

Devices must be able to communicate wirelessly using the network. The range of communication must be sufficiently large that devices can be placed anywhere in an average home. The smart home devices will need to be capable of receiving and transmitting data using this network with enough range.

The distance between nodes in our system must be no more than 50 meters. The will allow for any protocol to communicate with the necessary nodes.

System Responsiveness

The system must readily adapt to environmental changes to be effective. When in learning mode, the system does not make any decisions and therefore has no responsiveness requirement. However, when the system enters playback mode it must make decisions as fast as environmental changes are received. This will ensure that the system is as responsive as possible when a user performs an action.

Security Requirements

Device Connection Security

All commands to devices must be authenticated to ensure that they are from an authorized source. This is in order to eliminate the possibility of malicious entities taking control of a home's devices.

Remote Interface Security

Digital access to the hub's configuration interface must be secured using TLS 1.2 (RFC 5246) and HTTP basic authentication as described in RFC 2617. Use of these Internet Official Protocol Standards ensures that the system uses widely accepted authentication practices.

Quality Requirements

Learning Hub Reliability

The learning hub is the center of communications and is responsible for interfacing with the system user. It must record data on some form of internal storage to log actions that have occurred as well as decisions that is has made. It is critical that the learning hub not lose its data as this would set the system back to its initial, untrained state. Precautions should be taken so that in the case of a system failure or power failure, the critical system data is preserved. Hub operations should be atomic and reversible should they fail.

The learning hub must also be online and available to record system events. If the learning hub is to go into a state faulty state then it should indicate this to the user. The system must provide a mechanism for resetting itself if errors are occurring.

Device Reliability

Devices in the system do not need to meet as high of a standard as the learning hub for reliability. However, devices should have some indicator when faults occur. Devices may be hard to access so any device that can be restarted should provide functionality for doing so from the remote interface. If this is not possible then as a minimum, the remote interface should indicate whether or not a device has encountered a fault or if it is no longer responding. Restarting a device in the network should then reconnect to the system and retain all of its history within the learning hub.

3.2 Scenarios

In order to better understand the motivations for using the learning home automation system, we have created a number of scenarios where we expect the behaviour of homeowners to follow patterns that a machine learning algorithm could pick up on. These particular scenarios were chosen because they involve relationships between multiple devices which may not be obvious. The scenarios also include a list of smart devices involved. Using the devices identified in these scenarios, the team can compile a list of devices which will showcase the machine learning capabilities of the system for testing and demos.

3.2.1 Music Automation

Background

Smart home automation should make your life easy and fun. Imagine a group of people arrive at your house for a party. Your home automation system has learned how to set up your environment to give you the best experience possible. The lights dim, the temperature goes down, and the music goes up. Your home is now ready for your guests!

System Interaction

The home automation system will be able to interact with a music system. The remote interface will allow the user to turn on and off music and control the system volume. The system may also be able to control the specific speakers that are active and playing music.

The user may choose to schedule the operation of music within the home to start at a certain date or time. Alternatively, the user may train the system that when many people enter the home then music should start playing at a certain volume. This could be accomplished by entering training mode, having a large number of people enter the room and then turning on music. Motion sensors would be required to measure the occupancy of the home to enable this learning.

Using a similar method to the music training, the user could train the system to tune the temperature, lights, and any other devices they wish. This could indicate to the system that when there are many people in the home, all of the trained systems should be activated.

System Requirements

To be able to monitor the home, the following sensors may be of interest. These sensors will be use to monitor the occupancy of the home as well as determine the noise level of the home to appropriately adjust the music.

Sensor	Usage
Beam Motion Sensor	2 Beam sensors in series can be used to estimate occupancy
Audio Sensor	Audio sensor for tuning music volume to the atmosphere
PIR Motion Sensor	A PIR motion sensor can detect activity in a room

The system will require some devices to be able to produce the desired outputs. The following table lists example devices to allow this system to perform the tasks outlined in this scenario.

Device	Usage
Speakers	Controllable speakers for home configuration
Controllable Lights	Lights that can be controlled by the central system
Thermostat	Allows temperature to be adjusted

Sample Sequence

Setup

The home's main entrance is equipped with two break-beam motion sensors in series. The devices are configured so that they provide an estimate of the number of people in the house (OccupancyCount). Each of the main rooms contains an audio sensor (DenAudio, BasementAudio), and a sound system (DenSpeakers, BasementSpeakers). Each room also contains a motion sensor (DenMotion, BasementMotion). The basement is also set up with a strip of coloured smart lights.

Training Sequence

With their central hub in training mode, the homeowner arrives at the house with a car full of people. As each person enters the home, the motion sensors at the door notify the central hub that the number of people in the house is increasing. The party moves into the den and turns on some music. DenMotion reports the activity in the den to the central hub. The homeowner lowers the thermostat temperature so that the increased number of visitors doesn't make the house uncomfortable. As the level of conversation increases in the den, the homeowner decides to turn down the music to avoid noise complaints from the neighbour. Throughout the party, DenAudio reports changes in noise level to the central hub. When the homeowner changes the volume of the den speakers, DenSpeaker reports the change in its output to the central hub.

After an hour, half of the party moves to the basement to play some Scrabble. BasementMotion reports the new activity in the basement to the central hub. The Scrabble players decide to turn on some music to get everyone in the mood. BasementSpeakers reports the change in its state to the central hub. The homeowner decides to show off his light strips to his friends, and sets them to a dim red colour to enhance the party ambiance. The light strips report their colour and brightness level to the central hub (BasementLightStrip).

At the end of the night, everyone goes home. All music in the house turns off, the basement lightstrips turn off, and the thermostat is returned to its normal setting.

Expected Inputs

The log for this scenario is shown below. This log demonstrates the interactions that would be logged.

```
2017-01-01 19:00:00 OccupancyCount: 1
2017-01-01 19:00:01 OccupancyCount: 2
2017-01-01 19:00:02 OccupancyCount: 3
2017-01-01 19:00:03 OccupancyCount: 4
...
2017-01-01 19:05:00 OccupancyCount: 10
2017-01-01 19:06:50 MotionDen: Motion Detected
2017-01-01 19:10:00 DenAudio: Sound level 100
2017-01-01 19:10:30 DenSpeakers: Volume 80%
2017-01-01 19:11:00 DenAudio: Sound level 95
2017-01-01 19:11:15 Thermostat: Temperature 17 C
2017-01-01 19:11:30 DenAudio: Sound level 100
...
2017-01-01 20:00:30 DenAudio: Sound level 200
2017-01-01 20:05:00 DenSpeakeκ: Volume 70%
2017-01-01 20:05:30 DenAudio: Sound level 150
...
2017-01-01 20:29:50 MotionBasement: Motion Detected
2017-01-01 20:29:55 BasementSpeaker: Volume 60%
2017-01-01 20:30:00 BasementLightStrips: Colour red
2017-01-01 20:30:00 BasementLightStrips: Brightness 50%
...

2017-01-02 01:00:00 OccupancyCount: 2
2017-01-02 01:02:00 BasementMotion: No activity
2017-01-02 01:03:10 DenMotion: No activity
2017-01-02 01:03:15 DenAudio: Sound level 10
2017-01-02 01:03:16 BasementSpeaker: Volume 0%
2017-01-02 01:03:17 BasementLightStrips: Off
2017-01-02 01:03:18 DenSpeaker: Volume 0%
2017-01-02 01:05:00 Thermostat: 20 C
...
```

Expected Behaviour

The next time the homeowner has a Scrabble party, 10 guests arrive at the home. After they step through the door and motion is detected in the den, the den sound system turns on and adjusts its volume to an appropriate level. The thermostat lowers the temperature of the room to 20 degrees. The system continues to adjust the volume of the sound system to match the amount of noise detected in the room.

A group of the partygoers decide to move down to the basement for their game. When motion is detected in the basement, the system turns on the lightstrips to a dim red colour and turns the basement sound system on.

After everyone leaves for the night, BasementLightStrips, BasementSpeaker, and DenSpeakers turn off. Thermostat is re-adjusted to the temperature before the party began.

3.2.2 Efficient Lights and Temperature

Background

A smart home should reduce your energy bills and keep you comfy. During your work week, your home is left to cool during the day when no one is home. In the evening, before you arrive, the system heats the house to a comfortable temperature. As you arrive home, the lights automatically turn on in the rooms that you will enter. Later in the evening, the system cools the house to a comfortable sleeping temperature and dims the lights.

On the weekend, the house remains warm during the day while you are home. If you leave then to go to a store, the system turns off all the lights and lowers the temperature. When your arrive home again the system turns the lights back on and raises the temperature.

In the summer months, when it is more light outside, the system does not turn the home's lights on until later. In the winter months, the home turns the lights on earlier.

System Interaction

The system will need to interact with multiple sensors as well as light and temperature controllers. The remote interface will need to be able to display the state of all the sensors in the system. The remote will also need to offer control of the other devices in the system.

The system will also be able to be trained to obtain the desired output. To be able to have the lights turn off when the user leaves the room, the user could enter training mode with the lights on, leave the room and then turn off the lights. If this interaction was repeated then the system might learn this behaviour.

For the system to learn the desired temperature that the user desired, the learning process may be much longer. At different times of day the user will change the temperature. As environmental factors changes, the system will make these observations and use them to decide what the should be set to.

System Requirements

To enable light and temperature control, sensors will be needed to observe the system. The sensors will be needed to observe the ambient light and temperature of the home. There will also need to be sensors to determine the occupancy of the home. The following is a list of sensors that will be needed for this scenario.

Sensor	Usage
Light Sensor	Used to determine the amount of light in the home
Temperature Sensor	Need to observe the temperature of the home
Motion Sensor	Will provide information about the occupancy of the home

To enable this scenario, the system will need to be able to control a number of devices. The system will need to have control of the home's lights and thermostat. In addition to having the devices in the system, the system's user interface will need to provide control mechanisms for the device. The following is a list of the devices that will be needed and how they will be used in the system.

Device	Usage
Smart Lights	Lights that can be controlled through an API
Thermostat Controller	A device that can control the temperature through an API

Sample Sequence

Setup

The home is setup with multiple smart lights (SmartLights) for each room and a smart thermostat (SmartThermostat). Motion sensors are added to the main rooms (MotionBedroom, MotionKitchen, MotionDen) in the home to detect presence. Finally, to detect the ambient light per room, light sensors are added in conjunction with the motion sensors (LightBedroom, LightKitchen, LightDen).

Training Sequence

The user starts the day by turning on the lights in the bedroom at 7:00:00 AM. LightBedroom sends a notification to the central hub that the light levels have changed in the room and MotionBedroom sends a motion message to the hub. The hub logs these interactions at 7:00:25 AM.

The user then goes to the den to change the temperature to 22°C at 7:12:00 AM. On the way, the user turns on the den lights. MotionBedroom and MotionDen track the movement and send it to the hub. LightDen sends a message to the hub that it has been turned on and SmartTemperature sends the user's input to the hub as well. The hub receives the motion messages at 7:11:40 AM and 7:11:50 AM followed by the den lights at 7:12:05 AM and the temperature change at 7:12:25 AM.

The user then returns to the bedroom and prepares to leave for the day. MotionBedroom again sends a notification that the user is moving in the bedroom from 7:14:00 AM to 7:30:00 AM. The user then leaves the bedroom and turn the light off on the way out. MotionBedroom stops sending motion requests and LightBedroom sends a message that it has been turned off. The user then heads to the kitchen and turns on the lights to make breakfast. MotionKitchen and LightKitchen send messages to the hub that they have been activated at 7:30:25 AM and 7:30:40 AM. MotionKitchen continues to send motion events from 7:31:00 AM to 7:45:00 AM.

Once the user is done breakfast they are ready to leave for the day. The user turns off the lights in the kitchen and walks to the den. MotionKitchen stops sending messages and LightKitchen signals that it has been turned off. In the den, the user lowers the temperature back to 20°C and turns off the den lights before leaving. MotionDen beings sending messages at 7:45:25 AM while SmartThermostat notifies that it has been changed. As the user leaves, MotionDen stops sending motion notifications and LightDen indicates that it has been turned off.

Alternate Sequence

The user has the option to program all of the smart devices through the web user interface on a predefined schedule. They can choose to select the desired temperatures throughout the day as well as when the lights should turn off and on. Optionally, the user can manually set values through the web UI in real-time without a pre-configured schedule. These events will be logged as if the user was manually interacting with the physical device.

Expected Inputs

The log for this scenario is shown below. This log demonstrates the interactions that would be logged. Please not that duplicate messages have been omitted and replaced with `...`.

```
2016-10-08 7:00:00 MotionBedroom: Motion Detected
2016-10-08 7:00:25 LightBedroom: Light On
2016-10-08 7:11:40 MotionBedroom: Motion Detected
2016-10-08 7:11:50 MotionDen: Motion Detcted
2016-10-08 7:12:05 LightDen: Light On
2016-10-08 7:12:25 SmartThermostat: Change Temperature 22°C
2016-10-08 7:14:00 MotionBedroom: Motion Detected
...
2016-10-08 7:29:50 MotionBedroom: Motion Detected
2016-10-08 7:29:55 LightBedroom: Light Off
2016-10-08 7:30:00 MotionBedroom: Motion Detected
2016-10-08 7:30:25 MotionKitchen: Motion Detected
2016-10-08 7:30:40 LightKitchen: Light On
2016-10-08 7:31:00 MotionKitchen: Motion Detected
...
2016-10-08 7:44:50 MotionKitchen: Motion Detected
2016-10-08 7:44:55 LightKitchen: Light Off
2016-10-08 7:45:00 MotionKitchen: Motion Detected
2016-10-08 7:45:10 MotionDen: Motion Detected
2016-10-08 7:45:25 SmartThemostate: Change Temperature 22°C
2016-10-08 7:45:30 MotionDen: Motion Detected
2016-10-08 7:45:40 LightDen: Light Off
2016-10-08 7:45:45 MotionDen: Motion Detected
```

Expected Behaviour

The user wakes up and gets out of bed at 7:10:00 AM. MotionBedroom detects this motion and the hub notifies the LightBedroom turns on at 7:10:05 AM. The hub also notifies the SmartThermostat to change its temperature to 22°C at 7:10:10 AM. The user then gets ready for the day and leaves the bedroom at 7:38:00 AM and heads to the kitchen. MotionBedroom stops sending movement notifications at 7:38:00 AM and MotionKitchen starts at 7:38:15 AM. The hub notifies LightBedroom to turn off at 7:40:00 AM and LightKitchen to turn on at 7:38:20 AM.

The user makes and eats breakfast and leaves at 7:48:00 AM. All motion sensors stop logging motion. The hub turns off LightKitchen and sets the SmartThermostat to 20°C at 7:50:00 AM.

The system will continue to pick up patterns as it learns. The system may begin to relate days of the week to certain events and rely less on the motion sensors. If the user was to consistently wake up at 7:00 AM and turn the lights on every Monday then the system may always turn the lights on at that time on Monday regardless of motion. These other factors will need to be considered while designing this system.

3.2.3 Coffee Automation

Background

Routine tasks done on a periodic schedule and can be automated by a smart process. Every morning you wake up and make a pot of coffee before you go about your day. Making a pot of coffee is a task that can be handled automatically by the smart home system. The system should learn when you wake up and make your coffee for you.

Let's imagine that on the weekend you don't make any morning coffee, the system should learn this behaviour and adapt during the days of the week. On a day that you are not at home, the system should not make any coffee either.

System Interaction

The system will need to be able to interact with a number of sensors to detect the user's presence. The system will also need to be able to communicate to a smart coffee maker so that it can observe when it is running as well as turn it off and on. The system will also need to be able to differentiate between the different days of the week and the time of day.

In order to train the system, the user could put the system into training mode and then get into bed. The user could then get out of bed and go directly to the kitchen and make a pot of coffee. The system could observe the user's leaving the bed with motion sensors and track that they are making coffee in a smart coffee maker.

System Requirements

To track the user's motion in the home, the system will need motion sensors. To be able to differentiate between the days of the week and time, the system will also need access to a clock and a calendar. The following is a list of sensors that will be required for this interaction.

Sensor	Usage
Motion Sensors	Used to track user movement throughout the home
Clock	Used to determine the time of day that the user makes coffee
Calendar	Used to determine what day of the week the user makes coffee

To be able to actually make the coffee, a smart coffee maker will be needed. This is the only smart device that will be required to automate this scenario.

Sample Sequence

Setup

The home is setup with a motion sensor outside the user's bedroom (MotionBedroom), a motion sensor outside the bathroom (MotionBathroom), a motion sensor outside the kitchen (MotionKitchen) and, a smart coffee machine in the kitchen (SmartCoffee).

Training Sequence

The user wakes leaves their bedroom and heads towards the bathroom at 7:00:00 AM. MotionBedroom sends a motion detected message to the hub. The hub logs that MotionBedroom detected motion at 7:00:00 AM. MotionBathroom then sends a motion detected message to the hub. The hub then logs that MotionBathroom detected motion at 7:00:25 AM.

The user then leaves the bathroom and heads back to their bedroom. MotionBathroom sends a motion detected message to the hub. The hub logs MotionBathroom detected motion at 7:15:00 AM. MotionBedroom then sends a motion detected message to the hub. The hub logs MotionBedroom detected motion at 7:15:25 AM.

The user then heads from their bedroom to the kitchen. MotionBedroom sends a motion detected message to the hub. The hub logs MotionBedroom detected motion at 7:20:00 AM. MotionKitchen sends a motion detected message to the hub and the hub logs MotionKitchen detected motion at 7:21:00 AM. The user then turns on the coffee maker. Coffee sends a start brewing message to the hub at 7:21:30 AM. When the coffee is done, SmartCoffee sends a done message to the hub. The hub logs that SmartCoffee finished at 7:27:30 AM.

Alternate Sequences

There are a few different ways that the coffee maker could be setup to make coffee instead of being turned on manually. It could be scheduled to start at a specific time or it could be started by the user when they get up through the web UI. The difference in the logs for both these cases is simply when the log for the coffee starting and coffee done appear.

Expected Inputs

The logs for the training scenario would look like the following:

```
2016-11-04 7:00:00 MotionBedroom: Motion Detected
2016-11-04 7:00:25 MotionBathroom: Motion Detected
2016-11-04 7:15:00 MotionBathroom: Motion Detected
2016-11-04 7:15:25 MotionBedroom: Motion Detected
2016-11-04 7:20:00 MotionBedroom: Motion Detected
2016-11-04 7:21:00 MotionKitchen: Motion Detected
2016-11-04 7:21:30 SmartCoffee: Starting
2016-11-04 7:27:30 SmartCoffee: Done
```

This sequence would occur every weekday possibly with the coffee maker being scheduled on one day and the user starting the coffee maker from the web UI. A sample 5 days of logs could look like the following:

```
2016-11-07 7:00:00 MotionBedroom: Motion Detected
2016-11-07 7:00:25 MotionBathroom: Motion Detected
2016-11-07 7:15:00 MotionBathroom: Motion Detected
2016-11-07 7:15:25 MotionBedroom: Motion Detected
2016-11-07 7:20:00 MotionBedroom: Motion Detected
2016-11-07 7:21:00 MotionKitchen: Motion Detected
2016-11-07 7:21:30 SmartCoffee: Starting
2016-11-07 7:27:30 SmartCoffee: Done

2016-11-08 7:05:00 MotionBedroom: Motion Detected
2016-11-08 7:05:30 MotionBathroom: Motion Detected
2016-11-08 7:20:00 MotionBathroom: Motion Detected
2016-11-08 7:20:25 MotionBedroom: Motion Detected
2016-11-08-7:23:00 WebUI: Start Coffee requested
2016-11-08-7:23:10 SmartCoffee: Starting
2016-11-08 7:29:30 SmartCoffee: Done
2016-11-08 7:30:00 MotionBedroom: Motion Detected
2016-11-08 7:31:00 MotionKitchen: Motion Detected

2016-11-09 7:02:00 MotionBedroom: Motion Detected
2016-11-09 7:02:30 MotionBathroom: Motion Detected
2016-11-09 7:12:00 MotionBathroom: Motion Detected
2016-11-09 7:12:25 MotionBedroom: Motion Detected
2016-11-09 7:15:00 MotionBedroom: Motion Detected
2016-11-09 7:16:00 MotionKitchen: Motion Detected
2016-11-09 7:16:30 SmartCoffee: Starting
2016-11-09 7:22:30 SmartCoffee: Done

2016-11-10 7:00:00 MotionBedroom: Motion Detected
2016-11-10 7:00:30 MotionBathroom: Motion Detected
2016-11-10 7:12:00 Hub: Starting Scheduled Coffee
2016-11-10 7:12:02 SmartCoffee: Starting
2016-11-10 7:16:00 MotionBathroom: Motion Detected
2016-11-10 7:16:25 MotionBedroom: Motion Detected
2016-11-10 7:18:00 SmartCoffee: Done
2016-11-10 7:20:00 MotionBedroom: Motion Detected
```

```
2016-11-10 7:21:00 MotionKitchen: Motion Detected

2016-11-11 7:02:00 MotionBedroom: Motion Detected
2016-11-11 7:02:30 MotionBathroom: Motion Detected
2016-11-11 7:20:00 MotionBathroom: Motion Detected
2016-11-11 7:20:25 MotionBedroom: Motion Detected
2016-11-11 7:15:00 MotionBedroom: Motion Detected
2016-11-11 7:16:00 MotionKitchen: Motion Detected
2016-11-11 7:16:30 SmartCoffee: Starting
2016-11-11 7:22:30 SmartCoffee: Done
```

Expected Behaviour

From the logs the Aria system should be able to tell that the user's morning consists of going to the bathroom, going back to their room then going to the kitchen and having coffee. The system can determine that the average time it takes the user for waking up until they get to the kitchen is 19 minutes and a pot of coffee takes 6 minutes to make. Therefore it is expected when the system is in playback mode that it would start the coffee maker 13 minutes after it sees that MotionBedroom and MotionBathroom have been triggered if it is around 7:00 AM.

As the system gains more and more information there could be some other patterns that the system notices. For example it could be more accurate to only consider other mornings for the current day of the week when calculating the average time to kitchen. For example Wednesdays the user takes less time in the bathroom so there average time to kitchen on Wednesdays is closer to 15 minutes. The system should then on Wednesdays start the coffee maker 9 minutes after the user gets up. There could also be factors from time of year. The users routine could change depending on the month/season these are factors we need to consider while designing the learning algorithm.

3.2.4 Effortless Home Lighting

Background

A learning smart home should reduce the need to manually perform everyday environmental control tasks. The system should be able to take in information from multiple sensors and devices, and be able to make adjustments to the home environment based on that information.

The homeowner enters their home office on a bright afternoon, and the lights remain turned off. They continue working through the afternoon and into the evening. As the sun begins to set, the lights in the room turn on at a low intensity to maintain the current level of brightness. The lights continue to increase in brightness as the sun continues to set, without any interaction from the person in the room. When the homeowner exits the room, the lights shut off.

At the same time the following day, the home owner re-enters the office. The weather outside is dark and rainy. The lights turn on to a comfortable brightness level upon entry, and remain there as the home owner works into the evening again. Upon exiting the room, the lights shut off again.

System Interaction

The system is required to combine information about the external environment with information about the homes internal environment. Simply tracking any one factor will not result in the system being able to perform this scenario, as the required device output does not correspond linerally to any one input.

Having the external environment be a factor in this scenario makes it hard to train the system to specifically do this, as you cannot easily control the external light levels. However, assuming the home owner keeps the light in the room at a certain level while the system is learning, the behaviour should be able to to be learned.

System Requirements

Two different sensors are needed for this scenario. A passive infrared sensor would be used to detect when there is someone present in a room. An ambient light sensor for inside the room will also be needed.

Sensor	Usage
Light	Used to determine the amount of light in a room
PIR	Will provide information about the occupancy of a room

To enable this scenario, the system will need to be able to change the brightness level of lights. The following is a list of the devices that will be needed and how they will be used in the system.

Device	Usage
Smart Lights	Lights that can be controlled through an API

Sample Sequence

Setup

The home is setup with a PIR in the home owner's office (PIROffice), a light sensor in the office (LightOffice), and dimmable smart lights in the office (SmartLights).

Training Sequence

The user enters their office at 2:30:00 PM on Wednesday afternoon. PIROffice sends a motion detected message to the hub. The hub logs that PIROffice detected motion at 2:30:00 PM. It is a sunny afternoon, so the home owner does not turn on the lights. The owner continues working uninterrupted until 7:45:00 PM, when the sun begins to set. The hub logs that LightOffice is reading a lower value due to the drop in light in the office. The owner turns the lights on dimly, to maintain the previous light level of the room, causing the hub to log that SmartLights turned on. As the light level of the room continues to drop in the room, the owner continues to raise the brightness of the lights. The hub continues to log this interaction. Eventually, it is dark outside and the lights are at full brightness in the room.

When the owner is done working, they get up, turn off the lights, and leave the room. The hub logs that PIROffice detected motion at 9:00:00, and logs that SmartLights turned off at 9:00:03.

Alternate Sequences

The light level of a room will not always be the same at a given time of day. The owner could enter their office at 2:30:00 PM on Wednesday afternoon, and turn the lights on immediately because it is rainy outside. The hub would log motion detected by PIROffice at 2:30:00, and then the value of LightOffice increasing to the desired value immediately following. This sequence would end the same way as the primary training sequence.

Expected Inputs

The logs for the training scenario would look like the following: PIROffice = 0 when there is no motion detected, 1 when it is detecting motion

```
Initial Sensor States
LightOffice: 10
SmartLights: 0
PIROffice: 0
```

```
2016-11-06 14:30:00 [Sensor Update] PIROffice: 1
2016-11-06 19:39:00 [Sensor Update] LightOffice: 9
2016-11-06 19:42:00 [Sensor Update] LightOffice: 8
2016-11-06 19:45:00 [Sensor Update] LightOffice: 7
2016-11-06 19:45:10 [User Action]   SmartLights: 3
2016-11-06 19:45:12 [Sensor Update] LightOffice: 10
2016-11-06 19:48:00 [Sensor Update] LightOffice: 9
2016-11-06 19:51:00 [Sensor Update] LightOffice: 8
2016-11-06 19:55:00 [Sensor Update] LightOffice: 7
2016-11-06 19:55:10 [User Action]   SmartLights: 6
2016-11-06 19:55:12 [Sensor Update] LightOffice: 10
2016-11-06 19:58:00 [Sensor Update] LightOffice: 9
2016-11-06 20:01:00 [Sensor Update] LightOffice: 8
2016-11-06 20:05:00 [Sensor Update] LightOffice: 7
2016-11-06 20:05:10 [User Action]   SmartLights: 10
2016-11-06 20:05:12 [Sensor Update] LightOffice: 10
2016-11-06 21:00:00 [Sensor Update] PIROffice: 1
2016-11-06 21:00:07 [User Action]   SmartLights: 0
2016-11-06 21:00:08 [Sensor Update] LightOffice: 0
```

The alternate sequence could look like this:

```
Initial Sensor States
LightOffice: 0
SmartLights: 0
PIROffice: 0
```

```
2016-11-06 14:30:00 [Sensor Update] PIROffice: 1
2016-11-06 14:30:03 [User Action]   SmartLights: 10
2016-11-06 14:30:06 [Sensor Update] LightOffice: 10
2016-11-06 21:00:00 [Sensor Update] PIROffice: 1
2016-11-06 21:00:07 [User Action]   SmartLights: 0
2016-11-06 21:00:08 [Sensor Update] LightOffice: 0
```

Expected Behaviour

From the logs the Aria system should be able to tell that the desired light level of the owners office when occupied is 10. The system should automatically adjust the brightness level of the Smart Lights to achieve this level of brightness while the home owner is in the office. Upon the owner leaving the office, the system should recognize that the lights must be turned off.

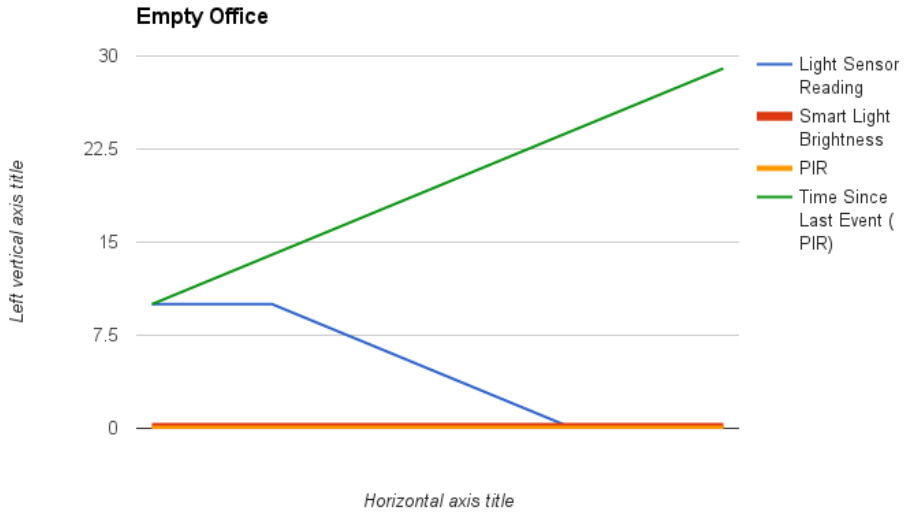
Sensor and Device Correlation

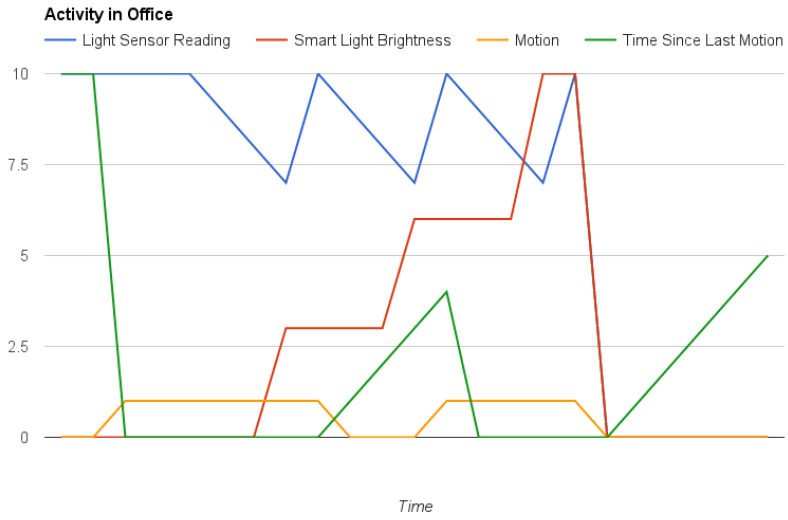
The graphs below depict the relationships between the light sensor, the motion sensor, and the smart light brightness level in multiple different scenarios. The goal of these graphs is to illustrate how the machine learning can interpret the sensor data to achieve the required functionality. The first relationship that the machine learning will need to recognize is the correlation between the PIR detecting motion and the lights being turned on. When no motion is detected, the light level detected by the LightOffice sensor is allowed to decrease to 0 without the lights being turned on.

Now that the system understands the relationship between no motion being detected and lights staying off, it needs to determine when it is appropriate to turn lights on when motion is detected. This is the relationship between the LightOffice sensor and the SmartLights brightness level. The first time the SmartLights are turned on by the homeowner is when the level of light detected by the LightOffice sensor falls below a certain level. The lights are turned on enough to maintain the level of brightness observed before it started decreasing. As the light levels continue to drop, the SmartLights brightness continues to rise in compensation. The threshold use to determine when the lights turn on is something that will be determined by how dark the homeowner allows the room to get before turning on the lights when training. The falling-rising pattern of the light level concludes when the SmartLights are at maximum brightness or when the light sensor is at a consistent level.

The final decision the system has to make is when it is appropriate to turn off the office lights. In the training scenario, the home owner turns the light off when leaving the room. The motion sensor stops being triggered around the same time that motion is no longer being detected. This has the potential to raise issues for the learning. For example, if the home owner is stationary for a short period of time in the room, the lights should not turn off. As well, if a new person enters the room and then leaves shortly after, the lights should not turn off. This is why the "Time Since Last Motion" (TSLM) variable is present. It is simply a timer that resets any time the motion sensor detects motion. Having this variable allows the absence of motion to be a value that is submitted to the machine learning algorithm. When the system is performing this scenario, the lights will likely remain on for a period of time after the homeowner leaves the room until the TSLM variable reaches a certain threshold. This threshold will be dependent on the sensitivity of the motion sensor. The more sensitive the sensor is to motion, the smaller the threshold will be. The reason for this is that if the sensor can detect motion such as the homeowners head moving while they are sitting at a computer typing, it will be easier to determine that they are still in the room. If the sensor is only able to notice large movement, this has the potential to lead to an inconvenience to the user in the form of the lights turning off while they are still present in the room. They would need to make a big enough motion to trigger the motion sensor, thereby turning on the lights and resetting the TSLM variable.

Graphs





Feature Extraction

- Time Since Last Motion

The length of time since motion had been last detected my the PIR. If no motion is detected for a long period of time, then the learning could interpret it as there being no occupancy in the room. This could be determined by using a counter on an interval. Whenever a message is received, this counter would be reset. If the counter reaches a certain amount of time then a message is logged saying no motion is detected. This feature alone is not sufficient to determine if there is a person in the room, as they could remain still for a long amount of time. Knowing the occupancy of the room would solve this issue.

- Occupancy of the Room

Having a system know the occupancy of a room is not a trivial task, and is one that this scenario is not currently equipped to be able to do. A possible solution would be to have a double break beam on a doorway. While not perfect, this would provide a reasonably accurate count of people entering and leaving a room. The occupancy of a room can be extracted from this data.

- Light Level Trend

Light sensors are generally noisy and will provide many samples over a short period of time. In order to have a more accurate representation of the state of the light in the room, the sensor reading's could be averaged to get a general state. This average value can then be compared to previous averages of the lights value to get the trend of the light in the room. This can be used to indicate if the light in the room is increasing or decreasing.

- Time of Day

The time of day will be relevant when determining the light information as it can help indicate the daylight available. It is likely that at the same time each day, the lights will be in the same state as the day before. Using the time of day can provide a good indication of the light levels that will be desired in the room.

3.3 Design

3.3.1 Introduction

Audience

This section describes the high level architecture of the Aria system. It details the major components within the system as well as their deployment artifacts. This section is intended for a technical audience to understand the organization, deployment and internals of the Aria system.

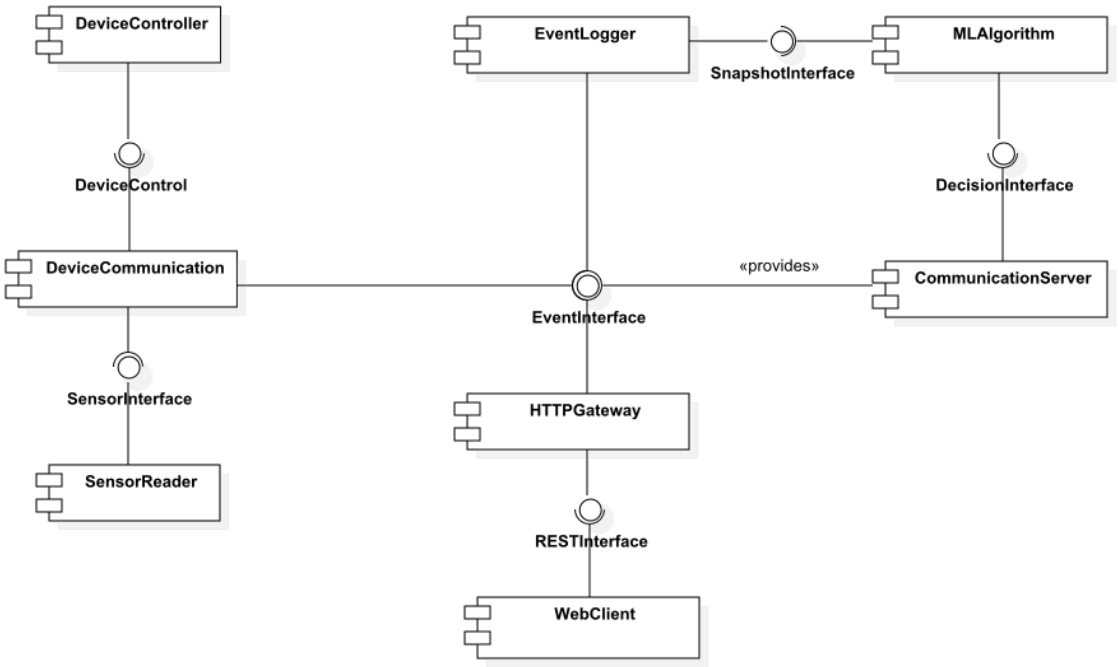
Background

The automated real-time interactive architecture (ARIA) system provides software for a machine learning smart home. The Aria system allows multiple smart devices to connect to a central hub to create a smart home. The system observes a user in a home environment and uses historical decisions to predict future actions.

This complex system is built upon several different subsystems that work together within a single hub. The hub communicates to several different devices over different protocols that are all synchronized with a custom Aria messaging protocol. This document outlines the details of these subsystems, interactions and the Aria messaging protocol.

3.3.2 System Components

Component Organization



Component Descriptions

Event Logger

The event logger is responsible for listening to all event signals and record them in a central database. This data will be accessible by the remote client as well as the ML algorithm component. The events recorded will be used to make decisions about actions to carry out. The event logger will consume the event interface that is provided by the communication server and will simply observe all data.

ML Algorithm

This component is responsible for observing events from devices, extracting features from the data, generating a model of interactions and then making decisions about actions to perform. The implementation of this algorithm can be customized as long as it provides the decision interface. The ML algorithm will receive data from the event logger using snapshots of data. These data snapshots will reduce the amount of information that the ML algorithm needs to process by removing redundant information.

Exchange Server

The communication server is responsible for routing all messages through the smart home system. The server has a cache of all connected devices and must store any related settings for each. Events and messages are routed through the communication server using the event interface. The communication server is also responsible for sending messages to other third party protocols. The server should provide a mechanism for installing plugins for other third party devices that are not natively supported.

HTTP Gateway

The HTTP gateway is responsible for supporting the web client interface. It must serve the web client's requests over a REST interface and translate them to the internal event interface used by the communication server. The gateway must be able to support multiple web clients.

Sensor Reader

The Sensor Reader is responsible for listening to a sensor and providing events when sensor data changes. The Sensor Reader will provide the Sensor Interface.

Device Controller

The Device Controller is responsible for controlling the physical device. It consumes events and modifies the output of the device accordingly. The Device Controller provides the Device Control Interface.

Device Control

This interface allows the device communication to pass events to the Device controller.

Component Interfaces

Event Interface

The hub must receive messages from several different devices; each device may provide a different interface for communication. In order for system components to process such messages, it is necessary to define a common interface for communicating with devices. Some examples of messages that must travel through the system are event messages from devices to signal a change in state, or a request from the REST API to change the state of a device. The Event Interface defines methods that a device or component must provide, and the data structures used to send and receive messages, regardless of the communication protocol that is used internally.

Decision Interface

The decision interface allows the ML algorithm to enter commands into the communication server which will be translated into events for the system. This interface must be provided by the ML algorithm and will be consumed by the communication server.

Snapshot Interface

This interface allows the ML algorithm to receive a subset of the data from the event logger. The snapshots remove any duplicate information before sending it to the ML algorithm for processing. This reduces the amount of data that needs to be sent as well as the amount of data that needs to be processed by the algorithm.

REST Interface

The REST interface allows web communication from web clients to the HTTP gateway. This representational state transfer protocol must allow the web client to carry out interactions with the various devices connected in the system as well as view the current system state. This interface must provide both a monitoring and controlling the system.

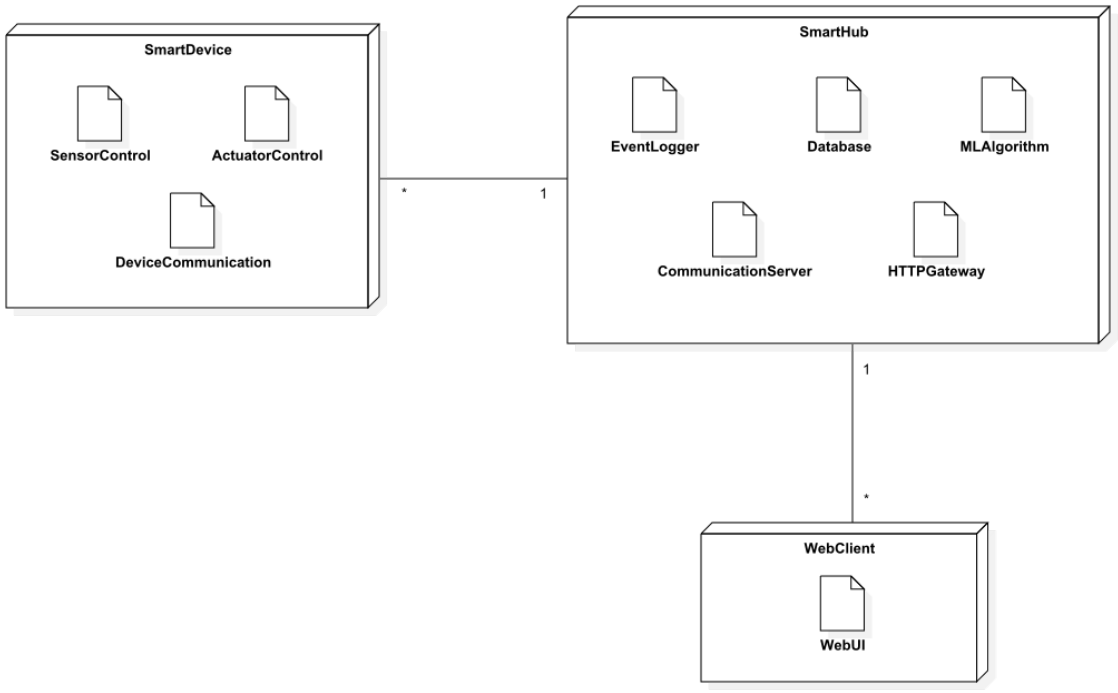
Device Communication

Device Communication is responsible for passing events between the device and the Communication Server. The Device Communication consumes the Device Control, Sensor Interface and the Event Interface.

Sensor Interface

This interface allows the Device Communication to read events from sensors and pass them on to the communication server. This interface is provided by the Sensor Reader and consumed by the Device Communication.

3.3.3 Deployment



Smart Hub

The central point of control of this smart home system is the smart hub. This hub is the central point of communication for all devices in the smart home system. It houses all of the data storage for events in the system and makes decisions using a smart learning algorithm. The hub will provide a minimal hardware interface for starting the system and changing the system state from training to normal to standby. The smart hub needs to be connected to a internet access point in order for it to serve the web interface to a client's computing device.

Smart Device

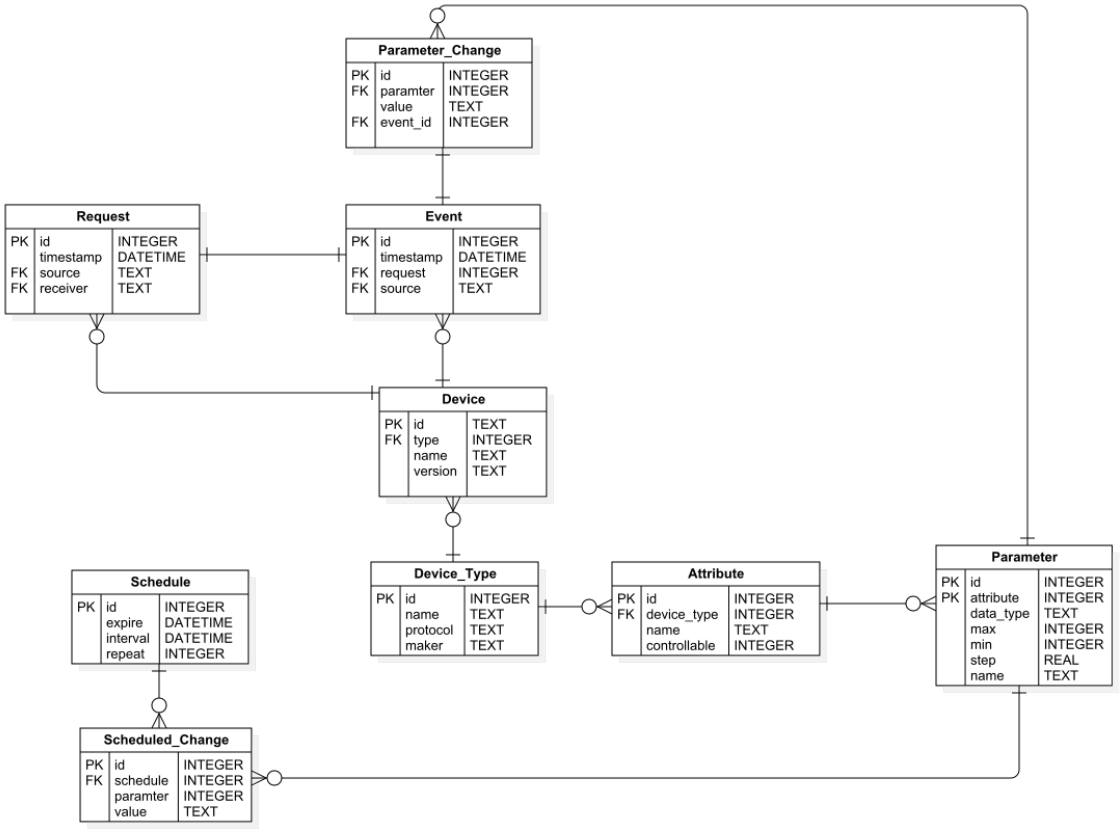
In this diagram, smart device refers to any smart device in the system. This could be a custom built device or a third party device. There will be many of these devices within the system all communicating to the central smart hub.

Web Client

The web client is the end user's browser and will present a remote interface for controlling the smart hub as well as all devices that are connected to the system. The web interface must be able to render on various industry standard browsers (Chrome, Firefox, IE, Safari).

3.3.4 Database

Database Entity-Relationship



Database Tables

Request

A request generated for a device. This request can be generated from the UI, the machine learning, or by an external user on the physical device. A row in the Request table is a requested state change of a controllable device.

Event

An event generated for the system. An event is a response to a request, changing the value of a device attribute based on the request. Events are linked to the Requests that cause them through the request_id foreign key. They are also linked to devices through the source foreign key, which is the device which caused the event. An event can also be generated by a non controllable sensor, in which case it will not be linked to a Request.

Parameter_Change

Holds the requested parameter to change or the parameter changed by an event, linked to by a foreign key in the Request and Device tables. The parameter field is a foreign to the Parameter table.

Devices

The individual devices connected to the system. Each device is uniquely identified by it's UUID, stored as the address. Each device has a device type, which can be used to determine what attributes are available for a device of that type.

Device Types

Specific manufacturer information for different types of possible devices (WeMo switch, Hue lights, etc...). This table also holds what communication protocol is needed to communicate with a device type and if it is an input or not.

Attributes

An attribute is a field of a device that provides information about the current device state. A controllable attribute is one such as the colour of a smart light bulb, where as a non-controllable attribute would be one such as reading from a sensor. Attributes can have multiple parameters associated to them. Take a smart light bulb for example; the attribute would be the colour of the bulb, and the parameters could involve different possible colours such as red, blue, and blue.

Parameters

Parameters are associated to an attribute. They are used to describe the possible states of their associated attribute.

Schedule_Change

Used to enable device scheduling. A parameter is associated with a value to change to, and a link to the schedule table. This link is used to determine when the requested parameter change should occur.

Schedule

A schedule determines when a parameter should change. It allows for a scheduled event to be recurring according to date, repeated for a set number of times, or simply occur once.

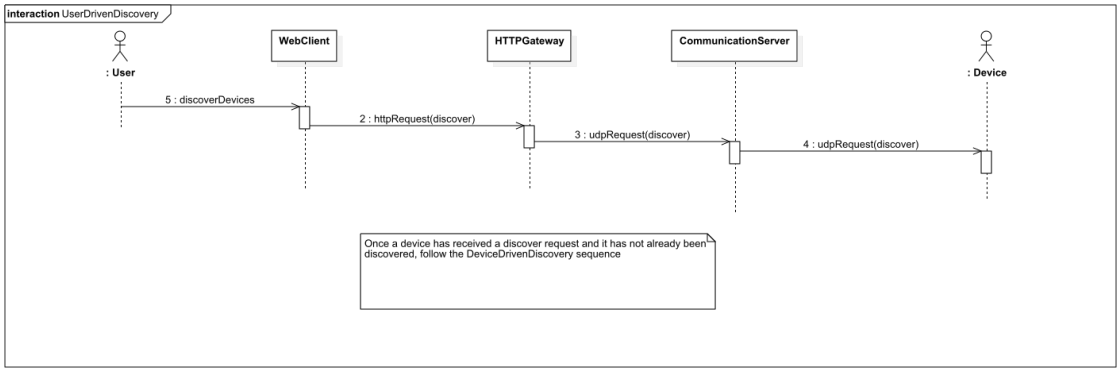
3.3.5 Discovery

Overview

In order to add a device to the central exchange hub registry, the device must be added through the discovery sequence. Until the device has been added, no communication can take place. Device discovery can be initiated from external devices or from the hub itself. This section outlines the discovery sequence in both scenarios.

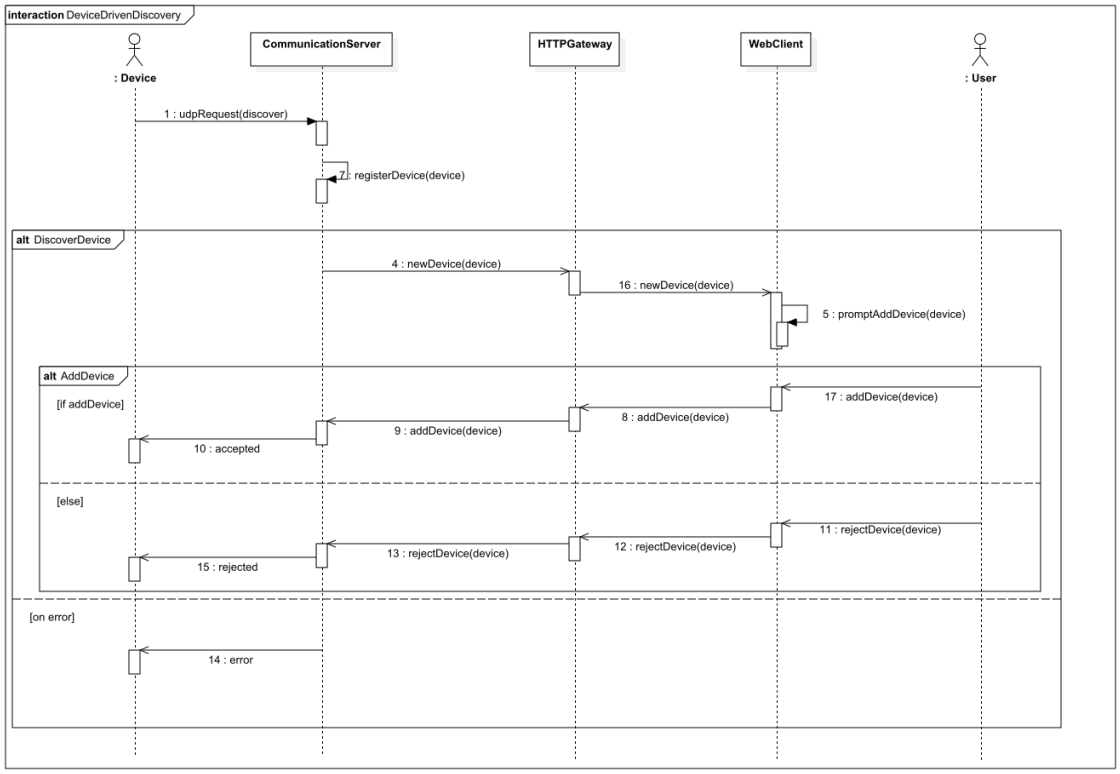
User Driven Discovery

User initiated discovery begins with the user requesting the server to send a discovery request. The message then propagates through the system to the communication server which sends a broadcast discovery message to find new devices. If a device receives a request to be discovered then it begins the device driven discovery sequence



Device Driven Discovery

Device initiated discovery begins with an initial discovery request sent from the device. The hub will receive a request for the discovery with device details. If the device has valid information then the request is propagated to the web client. Once the web client receives a notification that a new device is available, it prompts to user to add the device. The user can then choose to accept or reject the device which will send a message to the device.



3.3.6 Requests

Introduction

The purpose of this section is to provide an understanding of the interactions between an end user and the system for different types of requests.

Web Client

User

The User actor represents an end user of the system, which is the Web Client in this case. The user initiates all requests in the use case diagram for this system.

System Hub

The SystemHub actor represents the central hub of the ARIA system. The system hub is in charge of relaying information of the system to the User through the Web Client.

Device

The Device actor represents any device in the ARIA system. A device is responsible for relaying its specific information to the User through the Web Client.

List Devices

Return a list of all devices currently known by the system to the User.

View System Events

Return a log of recent events that have occurred in the system. An event is when the state of a device changes.

Get System Status

Return the status of the system to the user. This includes information such as what mode is the system currently operating in, current version number, number of connected devices, etc.

Set System Mode

Set the mode of the system. The different modes are: Standy(0x00), Normal(0x01), and Learning(0x02)

Add New Device

Add a new device to the system. This allows the end user to connect a new device to be controlled by the system and contribute to the machine learning.

Remove Device

Remove a device from the system. This allows a user to remove a device and its information, no longer allowing it to be controlled by the system. If added back to the system, there will be no stored information on it.

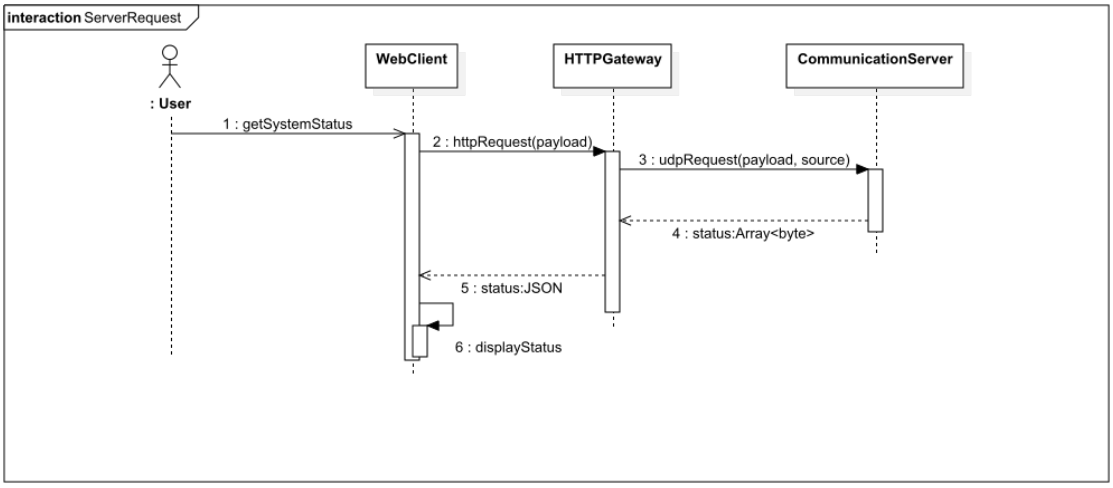
Get Device State

Return the current state of a device to a user. The state information could include if it is on, off, and other values depending on the device.

Set Device State

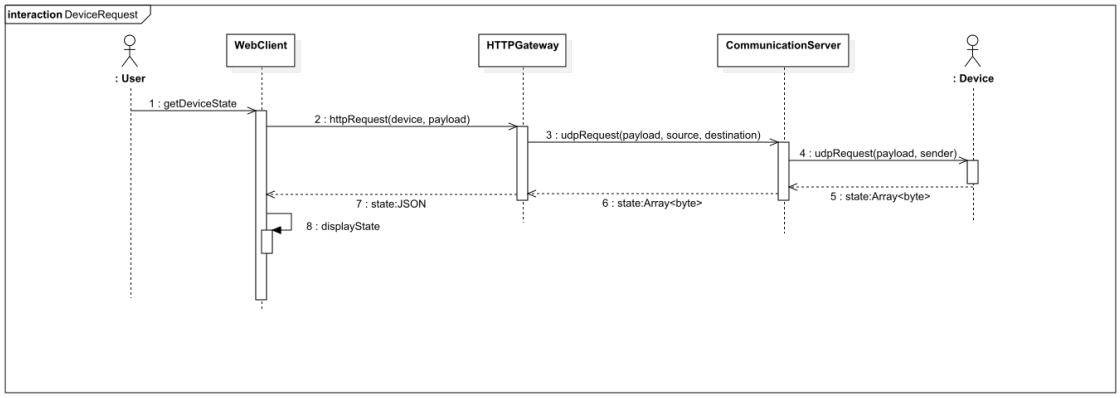
Set the current state of a device. The state information could include if it is on, off, and other values depending on the device.

Server Request



The Server Request diagram shows the workflow that occurs when an end user wants to retrieve information about the state of the communication server through the web client.

Device Request



The Device Request diagram shows the workflow that occurs when an end user wants to retrieve inform about a specific device in the system through the web client.

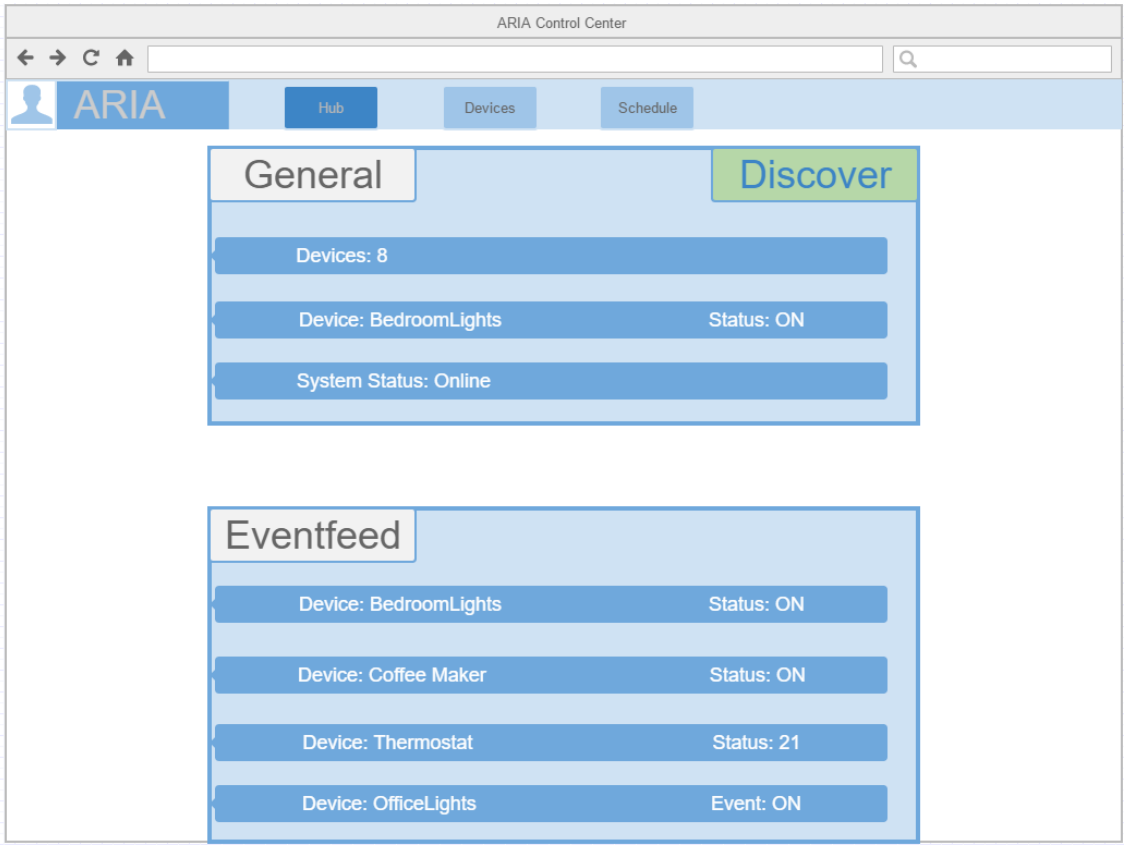
3.3.7 Remote User Interface

The user interface must be able to provide the user with control of the Aria system. It must provide features for observability and controllability of the system. This interface is the sole point of communication of the user to the system so it needs to be presented in a non-technical way that provides complete control.

In order to be accessible from any computer, the remote interface will be served as a web application to any standard browser. The user can log into their favourite browser and navigate to the remote application by typing in the address of their smart hub

Home Page

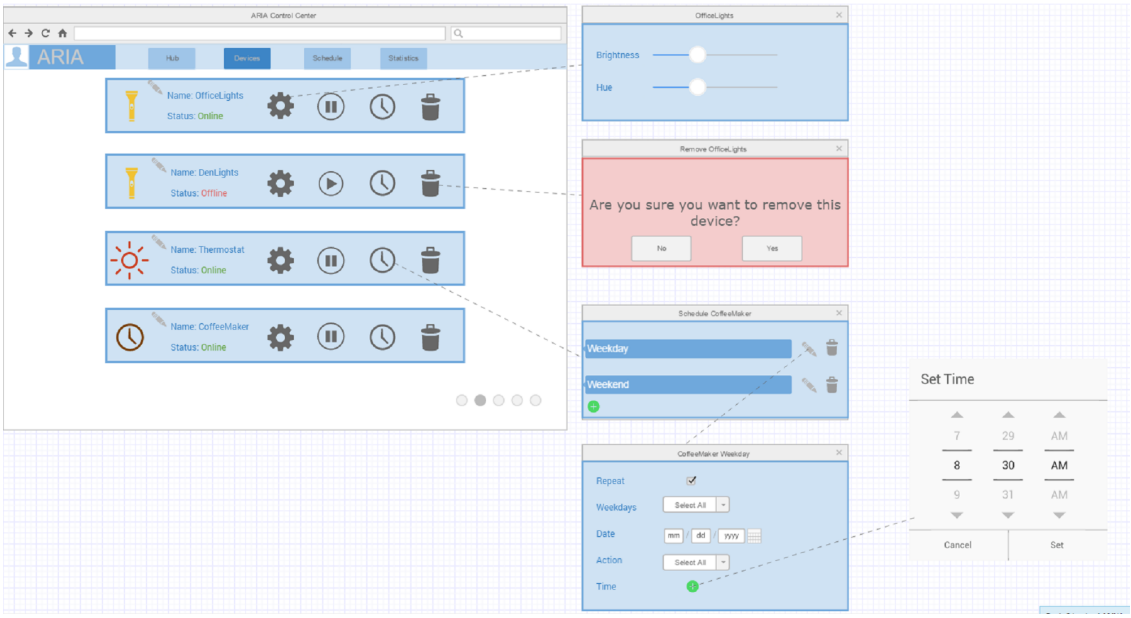
When the user initially logs into the smart hub page, they will be presented with the remote home screen. The starting page of the remote provides a general overview of the state of the system as well as an event feed of the latest device interactions. The user can use this page to change the system's state from standby to training or normal mode. For convenience, the user can jump right to device discovery and being configuring a new device in the system by clicking the "Discover" shortcut. Below is an outline of what the remote home page could look like.



Devices

In order to go into more details about each device, the user will also have a devices page. The devices page allows a user to control or configure a given device. The user can choose to perform a specific action by clicking the UI toggles for the appropriate task. Interacting with the UI in this manor to cause an event will be logged in the system in the same way as it would if the user had manually interacted.

Each device can also be scheduled to perform specific tasks at desired times. This can be done by pressing the schedule tab, adding an action and configuring the date and time. These scheduled tasks will override any smart hub decisions and will be top priority. Below is an outline of what the devices tab could look like.



Statistics

To aid in observability of the system, a statistics page will be provided. This page will allow the user to view all of the data logs that are contained within the system as well as plot them using various regressions through the UI. The plots created through the UI will help the user in understanding the decisions that the system makes by displaying its data in easily consumable graphs.

3.3.8 IPC Protocol

This section defines a basic protocol for interprocess communication (IPC) between the HTTP gateway and the central server.

Protocol Definition

The messages defined by this protocol are sent using UDP.

The protocol uses a simple request-response format. The server listens for messages on port 7600. All request packets must be acknowledged by a response packet. This protocol is intended for IPC purposes, so it is anticipated that both the server and client are running on the same machine and packet loss is extremely unlikely.

In the event that no response is received within a reasonable amount of time, the request should be re-sent.

Messages follow the general message structure shown below. Messages are encoded as bytes, using Big-Endian byte order.

General Message Structure

```
+--------+---------+----------+-------------+--------------+
|  type  |   size  |  sender  | destination |   payload    |
+--------+---------+----------+-------------+--------------+
| 1 byte | 4 bytes | 16 bytes |  16 bytes   | 'size' bytes |
+--------+---------+----------+-------------+--------------+
```

Message Types

The type field should contain one of the following values:

Name	Type	Value
Error	ERR	0x00
Discover	DISC	0x01
Request	REQ	0x02
Event	EVT	0x03
Response	RES	0x04

Data Types

This section defines the data structure format of data structures used in the response to requests

Data Type

Data type is an enum that is used to represent the data types of device attributes. Each data type has a unique set of behaviours and should be used appropriately. Each data type is configurable with three control parameters. The range of a data type can be controlled with a maximum (`max`) and minimum value (`min`). Some data types that do have a range of values should only increment in a specific sequence which can be controlled by the `step` parameter. The step of a value is the distance between two allowed values in a continuous range. It is a requirement that the step value is always a positive value and that the maximum value is greater than or equal to the minimum.

It should be noted that not all data types use these parameters. Some data types have pre-determined ranges or step values which are described for each data type below.

Binary

The `binary` type allows for control of a device from an 'On' and 'Off' state. These states are represented by 1 and 0 respectively. These are the only two values permitted by this data type.

Byte

The `byte` data type is an integer value with the range 0 to 255. This data type can be used to set a device in a specific mode or discrete value within this range. This data type ignores the max, min values of the data range as it is assumed the range is 0 to 255. The step value is respected by this data type but must be an integer value less than 255.

Integer

The `int` data type allows for a range of values from -2^31 to 2^31 - 1. This is appropriate for values that need to be discrete but must satisfy a specific range. All of the data types are respected by this data type.

Float

The `float` data type allows for a continuous value input to a device. This data type is represented by an IEEE-754 floating point number and the associated range. The maximum and minimum values for this data type should be integer values. The step for this data type can be a float value that is less that the maximum float value.

Color

A `color` data type refers to an RGB value. The data type is encoded as a 4-byte integer value where each byte represents a RGB value. The last byte is currently not used but is reserved as an alpha value if the requirement arises. For example, the value for white (`#FFFFFF`) would be encoded as `0xFF 0xFF 0xFF 0x00` or 4294967040 decimal.

Enumeration

An `enum` data type is used to represent a set of pre-determine values in a group. The values of the enumeration must be encoded with the data type in a special `values` field which can be an array of strings representing the enumerated values. The `value` field associated with an enumeration field will represent the zero based index of the value in the values array.

Time

The `time` data type is a millisecond representation of a time interval. The value of this data type is a millisecond value as an integer. The maximum and minimum parameters to this data type will be used to limit the range of time available. The step can also be used but must be an integer value.

Date

The `date` data type is a day representation of a time interval. The value of this data type is a day offset represented as an integer. All of the data type parameters can be used with this data type but they must be integer values.

String

The `string` data type is an open ended input which is represented by an arbitrary character sequence. The maximum and minimum parameters for this data type can be used to limit the length of the character sequence. The step parameter has no impact with this data type.

Attribute Parameter

Represents a parameter to an attribute of a device

```
{
    "name"      : <string>,
    "dataType"  : <DataType>,
    "max"       : <int>,
    "min"       : <int>,
    "step"      : <float>
}
```

- **dataType**: a DataType from the above section
- **max**: the max value the attribute can be, this may be null
- **min**: the min value the attribute can be, this may be null
- **name**: the attribute name, this is used in set and get request to query the device
- **step**: the difference between each acceptable value of the device, may be null

Attribute

Attributes represent the different attributes of a device that can be viewed and/or changed

```
{
    "name"              : <string>,
    "isControllable"    : <boolean>,
    "parameters"        : [<AttributeParameter>]
}
```

- **name**: Name of the attribute
- **isControllable**: boolean value which specifies whether the device is to be interpreted as a sensor or an output device

Device Type

The Device type represents the different types of devices that are in the system. The device type contains information about the manufacturer of the device the protocol the device speaks and the different attributes that the device has.

```
{
    "attributes"    : [<Attribute>],
    "maker"         : <string>,
    "name"          : <string>,
    "protocol"      : <string>
}
```

- **name**: the name of the type of device (WeMo Switch)
- **protocol**: specifies what adapter will be needed (Z-Wave, WeMo, etc)
- **maker**: device manufacturer name (Samsung, Aeon Labs, etc)

Device

A device is a representation of a device that is connected to the system and can provide input or be controlled

```
{
    "address"    : <string>,
    "deviceType" : <DeviceType>,
    "name"       : <string>,
    "version"    : <string>
```

```
}
```

- **address**: a unique UUID String used to identify a device
- **deviceType**: the DeviceType related to the device
- **name**: a user specified name for the device
- **version**: the device version number from the manufacturer

Requests

This section defines the format of the request and response payloads that are understood by the central server.

System Status

<table>
<tr><td>Title</td><td>Get Hub Status
Return the hub status and any properties associated with the hub</td></tr>
<tr><td>Destination</td><td>The message must be addressed to the hub's well known ID: <code>00000000-0000-0000-0000-000000000000</code></td></tr>
<tr><td>Message</td><td><pre>02 XX XX XX XX YY YY YY YY YY YY YY YY YY YY YY YY YY YY YY YY
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 PAYLOAD</pre>
<ul>
<li>XX: Indicates the bytes of the PAYLOAD length</li>
<li>YY: Indicates the bytes of the sender's address</li>
<li>PAYLOAD: The body of the message<pre>{
    "get" : "status"
}</pre></li>
</ul></td></tr>
<tr><td>Success Response</td><td>Below is an example of a successful response.<pre>04 XX XX XX XX 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
YY YY YY YY YY YY YY YY YY YY YY YY YY YY YY YY PAYLOAD</pre>
<ul>
<li>XX: Indicates the bytes of the PAYLOAD length</li>
<li>YY: Indicates the bytes of the original sender's address</li>
<li>PAYLOAD: The response to the status request<pre>{
    "response" : "status",
    "value"   : {
        "mode"   : 1,
        "name"   : "Smart Hub",
        "devices"   : 5,
        "version"   : "0.2.3"
    }
}</pre></li>
</ul></td></tr>
<tr><td>Error Response</td><td>Errors are returned as an error packet of type <code>0x00</code><pre>00 XX XX XX XX 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
YY YY YY YY YY YY YY YY YY YY YY YY YY YY YY YY PAYLOAD</pre>
<ul>
<li>XX: Indicates the bytes of the PAYLOAD length</li>
<li>YY: Indicates the bytes of the original sender's address</li>
<li>PAYLOAD: The response to the status request<pre>{
    "error" : "Invalid request"
}</pre></li>
</ul></td></tr>
</table>

Scan Devices

Title	**Start Device Discovery** Initiates a discovery sequence to find new devices on the network
Destination	The message must be addressed to the hub's well known ID: `00000000-0000-0000-0000-000000000000`

Message	`02 XX XX XX XX YY YY YY YY YY YY YY YY YY YY YY YY YY YY YY`
	`00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 PAYLOAD`
	• **XX**: Indicates the bytes of the PAYLOAD length
	• **YY**: Indicates the bytes of the sender's address
	• **PAYLOAD**: The body of the message
	`{ "action" : "discover" }`

Success Response	Below is an example of a successful response.
	`04 XX XX XX XX 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00`
	`YY YY YY YY YY YY YY YY YY YY YY YY YY YY YY YY PAYLOAD`
	• **XX**: Indicates the bytes of the PAYLOAD length
	• **YY**: Indicates the bytes of the original sender's address
	• **PAYLOAD**: Message reports if discovery has started successfully
	`{ "success" : true, }`

Error Response	Errors are returned as an error packet of type `0x00`
	`00 XX XX XX XX 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00`
	`YY YY YY YY YY YY YY YY YY YY YY YY YY YY YY YY PAYLOAD`
	• **XX**: Indicates the bytes of the PAYLOAD length
	• **YY**: Indicates the bytes of the original sender's address
	• **PAYLOAD**: The response to the status request
	`{ "error" : "Discovery failed reason" }`

List Devices

Title	**List Hub Devices** Returns the devices that are currently connected to the hub
Destination	The message must be addressed to the hub's well known ID: `00000000-0000-0000-0000-000000000000`

	`02 XX XX XX XX YY YY YY YY YY YY YY YY YY YY YY YY YY YY YY`
	`00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 PAYLOAD`
	• **XX**: Indicates the bytes of the PAYLOAD length

Message

- **YY**: Indicates the bytes of the sender's address
- **PAYLOAD**: The body of the message

```
{
    "get" : "devices"
}
```

Success Response

Below is an example of a successful response.

```
04 XX XX XX XX 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
YY YY YY YY YY YY YY YY YY YY YY YY YY YY YY YY PAYLOAD
```

- **XX**: Indicates the bytes of the PAYLOAD length
- **YY**: Indicates the bytes of the original sender's address
- **PAYLOAD**: The response to the status request

```
{
    "devices": [
        {
            "version"     : "1.2.1",
            "name"        : "Light Sensor",
            "address"     : "3c2538dd-64ed-4a0c-9ed3-14b2219feb11",
            "deviceType"  : {
                "name"          : "WeMo UPnP Light Sensor",
                "maker"         : "WeMo",
                "protocol"      : "UPnP",
                "attributes"    : [
                    {
                        "name"              : "State",
                        "isControllable"    : true,
                        "parameters"        : [
                            {
                                "name"      : "Hue",
                                "value"     : 79,
                                "dataType"  : "int",
                                "max"       : 255,
                                "min"       : 0,
                                "step"      : 1
                            },
                            ...
                        ]
                    },
                    ...
                ]
            }
        }
        ...
    ]
}
```

Error Response

Errors are returned as an error packet of type `0x00`

```
00 XX XX XX XX 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
YY YY YY YY YY YY YY YY YY YY YY YY YY YY YY YY PAYLOAD
```

- **XX**: Indicates the bytes of the PAYLOAD length
- **YY**: Indicates the bytes of the original sender's address
- **PAYLOAD**: The response to the status request

```
{
    "error" : "Invalid request"
}
```

Device Information

Title	**Request Device Status** Return the state of a device in the system
Destination	The message must be addressed to the device of interest

Message	`02 XX XX XX XX YY YY YY YY YY YY YY YY YY YY YY YY YY YY YY ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ PAYLOAD`
	• **XX**: Indicates the bytes of the PAYLOAD length
	• **YY**: Indicates the bytes of the sender's address
	• **ZZ**: Indicates the bytes of the device address
	• **PAYLOAD**: The body of the message
	`{ "get" : "status" }`

Success Response	Below is an example of a successful response.
	`04 XX XX XX XX ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ YY YY YY YY YY YY YY YY YY YY YY YY YY YY YY YY PAYLOAD`
	• **XX**: Indicates the bytes of the PAYLOAD length
	• **YY**: Indicates the bytes of the original sender's address
	• **ZZ**: Indicates the bytes of the original device address
	• **PAYLOAD**: The state of the device
	`{ "version" : "1.2.1", "name" : "Light Sensor", "address" : "3c2538dd-64ed-4a0c-9ed3-14b2219feb11", "deviceType" : { "name" : "WeMo UPnP Light Sensor", "maker" : "WeMo", "protocol" : "UPnP", "attributes" : [{ "name" : "State", "isControllable" : true, "parameters" : [{ "name" : "Hue", "value" : 79, "dataType" : "int", }, ...] }, ...] } }`

	Errors are returned as an error packet of type `0x00`. Note: this response example originates from the hub as there is no device with the given address `00 XX XX XX XX 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 YY YY YY YY YY YY YY YY YY YY YY YY YY YY YY YY PAYLOAD`

Error Response	• **XX**: Indicates the bytes of the PAYLOAD length • **YY**: Indicates the bytes of the original sender's address • **PAYLOAD**: The response to the status request `{ "error" : "Unknown device" }`

Control Device

Title	**Control Device Attribute** Changes the state of a device by passing parameters to a desired attribute
Destination	The message must be addressed to the device of interest

Message	`02 XX XX XX XX YY YY YY YY YY YY YY YY YY YY YY YY YY YY ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ PAYLOAD`
	• **XX**: Indicates the bytes of the PAYLOAD length
	• **YY**: Indicates the bytes of the sender's address
	• **ZZ**: Indicates the bytes of the device address
	• **PAYLOAD**: The body of the message
	`{ "set" : "attribute name" "value" : [ { "name" : "parameter 1", "value" : "new value" } ... ] }`

Success Response	Below is an example of a successful response.
	`03 XX XX XX XX ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ YY YY YY YY YY YY YY YY YY YY YY YY YY YY YY YY PAYLOAD`
	• **XX**: Indicates the bytes of the PAYLOAD length
	• **YY**: Indicates the bytes of the original sender's address
	• **ZZ**: Indicates the bytes of the original device address
	• **PAYLOAD**: If the device has been updated
	`{ "response" : "attribute name", "value" : { "device" : "Temperature Sensor", "deviceType" : "ZigBee Temperature Sensor", "attribute" : { "name" : "State", "parameters" : [ { "name" : "State", "value" : 30, "dataType" : "float" } ... ] }`

<table>
<tr><td></td><td>

```
            }
        }
    }
```

</td></tr>
<tr><td>Error Response</td><td>

Errors are returned as an error packet of type `0x00`.

```
00 XX XX XX XX ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ
YY YY YY YY YY YY YY YY YY YY YY YY YY YY PAYLOAD
```

- **XX**: Indicates the bytes of the PAYLOAD length
- **YY**: Indicates the bytes of the original sender's address
- **ZZ**: Indicates the bytes of the original device address
- **PAYLOAD**: The response to the control request

```
{
    "error" : "Bad parameter value 'parameter 1':'new value'"
}
```

</td></tr>
</table>

3.3.9 Database Interface

General

This interface provides a way for components in our system to query the database using the defined methods. Any component which has an instance of the Retriever class has access to these methods. A combination of prepared statements and having the database only accessed through an instance of Retriever eliminates the possibility of SQL injection, protecting our system.

Setup Instructions

To test this interface, there must be an existing database populated with event records. Accomplishing this through our system is a fairly comprehensive task. A database can be created by passing the desired name to the Database class. The created database will initially be empty. To populate it, devices must be created and registered to the hub. Events then need to be generated from these devices, sent through the system, and logged in the database. At this point, this interface is ready to be tested.

An alternative to this is to create and populate a test database manually. An instance of Retriever could be contained by a stubbed component and this interface could then be tested.

Interface Documentation

Event Window

Title	**Get Recent Events Across All Devices in a Window** Allows the retrival of a list of events starting from a specified index, ignoring events from specific devices if desired.
Data Params	• **start** : Index in the database to start retrieving from, with 0 being the most recent record • **count** : Number of events to retrieve • **ignore** : A list of device UUID's. Any events generated by a device with a UUID in this list will be ignored.
Sample Response	• **type** : List of dictionaries representing table records `[ { "id" : 2, "timestamp" : "2016-11-26 22:59:52", "request" : 3, "source" : "123e4567-e89b-12d3-a456-426655440000", "change" : { "id" : 4, "parameter" : { "id" : 2, "attribute" : 18, "data_type" : "color" }, "value" : "#F5F6AC" } } ... ]`
Sample Call	`getEventWindow(5, 10, [])`

Device Events

Title	**Get Recent Events for One Device in a Window** Allows the retrival of a list of events starting from a specified index, ignoring events from specific devices if desired.
Data Params	• **id** : UUID of device to get events from • **start** : Index in the database to start retrieving from, with 0 being the most recent record • **count** : Number of events to retrieve
Sample Response	• **type** : List of dictionaries representing table records `[ { "id" : 2, "timestamp" : "2016-11-26 22:59:52", "request" : 3, "source" : "123e4567-e89b-12d3-a456-426655440000", "change" : { "id" : 4, "parameter" : { "id" : 2, "attribute" : 18, "data_type" : "color" }, "value" : "#F5F6AC" } } ... ]`
Sample Call	`getDeviceEvents("123e4567-e89b-12d3-a456-426655440000", 10, 5)`

3.3.10 Gateway REST API

General

The gateway interface provides a public interface for controlling the Aria system using HTTP. The gateway uses a REST protocol for requesting or controlling static data about the system. For dynamic data, the gateway uses websocket messages. Below is the public REST API for the gateway. To see the available events over websockets see gateway websocket events.

Endpoint Documentation

Discovery

Title	**Start Device Discovery** This method initiates the discovery sequence for all adapters in the central hub. The return from this call will indicate if the process has started but not if any devices have yet been discovered. All device discoveries will be returned over websocket messages (see device discovered) for more details.
URL	`/hub/discover`
Method	**GET**
URL Params	**None**
Data Params	**None**
Success Response	**Example:** **Code:** `200` **Content:** `{ }`
Error Response	**Example:** **Code:** `500 Internal Server Error` **Content:** `{ error : "Unable to initiate discovery" }`
Sample Call	`curl -X GET http://localhost:8080/hub/discover`

Hub State

Title	**Get Hub State** Returns the current state of the hub. This will return the mode, number of connected devices and version of the hub.
URL	`/hub/state`
Method	`GET`
URL Params	**None**
Data Params	**None**
Success Response	**Example:** **Code:** `200` **Content:** `{ "version" : "1.0.0", "mode" : 1, "name" : "Smart Hub", "devices" : 8 }`
Error Response	**Example:** **Code:** `400 Bad Request` **Content:** `{ error : "Invalid Request" }`
Sample Call	`curl -X GET http://localhost:8080/hub/state`

Hub Mode

Title	**Get Hub Mode** Returns the mode of the hub
URL	`/hub/mode`
Method	`GET`
URL Params	**None**
Data Params	**None**
Success Response	**Example:** **Code:** `200` **Content:** `{ "mode" : 1 }`
Error Response	**Example:** **Code:** `400 Bad Request` **Content:** `{ error : "Invalid request" }`
Sample Call	`curl -X GET http://localhost:8080/hub/mode`

Title	**Set Hub Mode** Update the mode of the hub
URL	`/hub/mode`
Method	`POST`
URL Params	**None**
Data Params	• **mode**: The new mode of the hub (one of 0 = Standby, 1 = Normal, 2 = Learning) **Example:** Set mode to *Learning* **Content:** `{ "mode" : 2 }`
Success Response	**Example:** **Code:** `200` **Content:** `{ "mode" : 2 }`
Error Response	**Example:** **Code:** `400 Bad Request` **Content:** `{ error : "Invalid mode" }`

Sample Call	**Example:** Set the mode to *Normal*
	`curl -X POST http://localhost:8080/hub/mode \`
	`  -H 'Content-Type: application/json' \`
	`  --data '{ "mode" : 1}'`

Event Log

Title	**Get Event Log** Returns an event window from the hub of the requested window in reverse order (ensuring that the most recent event is first).
URL	`/hub/events`
Method	`POST`
URL Params	**None**
Data Params	• **start**: The index of the record to start at (0 indicates the most recent record). • **count**: The number of records to include in the window **Example:** Retrieve the last 10 events **Content:** `{ "start" : 0, "count" : 10 }`
Success Response	Upon successful response, the requested records will be returned. The records will be ordered from most recent to least recent. **Example:** **Code:** `200` **Content:** `{ "records" : [ { "index" : 10, "timestamp" : 1480262533722, "device" : "Temperature Sensor", "deviceType" : "ZigBee Temperature Sensor", "attribute" : { "name" : "State", "parameters" : [ { "name" : "State", "value" : 30, "dataType" : "float" } ... ] } } ... ] }`
Error Response	**Example:** **Code:** `500 Internal Server Error` **Content:** `{ error : "Invalid event window" }`
Sample Call	**Example:** Retrieve a window of 10 records starting after 10 `curl -X POST http://localhost:8080/hub/events \ -H 'Content-Type: application/json' \ --data '{ "start" : 10, "count" : 10 }'`

Device List

Title	**Get Devices** Returns a listing of all devices in the system
URL	`/device/list`
Method	`GET`
URL Params	**None**
Data Params	**None**
Success Response	Upon successful response, all devices will be returned. **Example:** **Code:** `200` **Content:** `{ "devices": [{ "version" : "1.2.1", "name" : "Light Sensor", "address" : "3c2538dd-64ed-4a0c-9ed3-14b2219feb11", "deviceType" : { "name" : "WeMo UPnP Light Sensor", "maker" : "WeMo", "protocol" : "UPnP", "attributes" : [{ "name" : "State", "isControllable" : true, "parameters" : [{ "name" : "Hue", "value" : 79, "dataType" : "int", "max" : 255, "min" : 0, "step" : 1 }, ...] }, ...] } } ...] }`
Error Response	**Example:** **Code:** `500 Internal Server Error` **Content:** `{ error : "Failed request" }`
Sample Call	`curl -X GET http://localhost:8080/device/list`

Device Events

Title	**Get Device Event Log** Returns an event window of logs from a specific device in reverse order (ensuring that the most recent event is first).
URL	`/device/:id/events`
Method	`POST`

URL Params	• **id**: The id of the device to request the logs for (ex. `3c2538dd-64ed-4a0c-9ed3-14b2219feb11`)
Data Params	• **start**: The index of the record to start at (0 indicates the most recent record). • **count**: The number of records to include in the window **Example:** Retrieve the last 10 events **Content:** `{ "start" : 0, "count" : 10 }`
Success Response	Upon successful response, the requested records will be returned. The records will be ordered from most recent to least recent. **Example:** **Code:** `200` **Content:** `{ "total" : 100, "records" : [ { "index" : 10, "timestamp" : 1480256989762, "device" : "Light Sensor", "deviceType" : "Aeon Labs UPnP Light Sensor", "attribute" : { "name" : "State", "parameters" : [ { "name" : "State", "value" : 69, "dataType" : "time", } ... ] } } ... ] }`
Error Response	**Example:** **Code:** `400 Bad Request` **Content:** `{ error : "Unknown device" }` OR **Example:** **Code:** `500 Internal Server Error` **Content:** `{ error : "Invalid event window" }`
Sample Call	**Example:** Retrieve a window of 10 records starting after 10 for device with id `3c2538dd-64ed-4a0c-9ed3-14b2219feb11` `curl http://localhost:8080/device/3c2538dd-64ed-4a0c-9ed3-14b2219feb11/events \ -X POST -H 'Content-Type: application/json' \ --data '{ "start" : 10, "count" : 10 }'`

Device Values

Title	**Set Device Attribute** Sets the value of an attribute of a device
URL	`/device/:id/setAttribute`
Method	`POST`
URL Params	• **id**: The id of the device to set the attribute for (ex. `3c2538dd-64ed-4a0c-9ed3-14b2219feb11`)
Data Params	• **name**: The name of the attribute to set • **value**: A list of parameters and their values for the attribute **Example:** Set the brightness of a light **Content:** `{ "name" : "brightness", "value" : [ { "naame" : "level", "value" : 50 } ... ] }`
Success Response	Upon successful response, value of the updated attribute will be returned **Example** { "response" : "brightness", "device" : "Light Bulb", "deviceType" : "Aeotec Dimmable LED", "attribute" : { "name" : "brightness", "parameters" : [{ "name" : "level", "value" : 50, "dataType" : "byte", } ...] } }
Error Response	**Example:** **Code:** `400 Bad Request` **Content:** `{ error : "Unknown device" }` OR **Example:** **Code:** `500 Internal Server Error` **Content:** `{ error : "Invalid data format" }`
Sample Call	**Example:** Set the brightness value of device with id `3c2538dd-64ed-4a0c-9ed3-14b2219feb11` `curl http://localhost:8080/device/3c2538dd-64ed-4a0c-9ed3-14b2219feb11/setAttribute \ -X POST -H 'Content-Type: application/json' \ --data '{ "name" : "brightness", "value" : ["level" : 50] }'`

3.3.11 Gateway Websocket Protocol

General

This websocket protocol provides push events for clients listening to the gateway events. These events notify when a device has been added or an event has occurred. They can be used to observe the system. Currently there is no plan for adding push back over the sockets for dynamic control.

Endpoint Documentation

Device Discovered

Title	**Event: Device Discovered** Triggered when the hub discovers a new device in the network. This event typically follows a request for discovery but can be manually initiated by adding a device to the network.
Event Name	`device.discovered`
Data Params	The callback value will be the device that is to be added. **Example**:

```
{
    "version"       : "1.2.1",
    "name"          : "Light Sensor",
    "address"       : "3c2538dd-64ed-4a0c-9ed3-14b2219feb11",
    "deviceType"    : {
        "name"          : "WeMo UPnP Light Sensor",
        "maker"         : "WeMo",
        "protocol"      : "UPnP",
        "attributes"    : [
            {
                "name"              : "State",
                "isControllable"    : true,
                "parameters"        : [
                    {
                        "name"      : "Hue",
                        "value"     : 79,
                        "dataType"  : "int",
                        "max"       : 255,
                        "min"       : 0,
                        "step"      : 1
                    },
                    ...
                ]
            },
            ...
        ]
    }
}
```

Device Event

Title	**Event: Device Event** Triggered any time a device changes state through the change in a parameter value
Event Name	`device.event`
Data Params	The callback value will be the device that is to be added. **Example**:

```
{
    "timestamp"     : 1480256989762,
    "device"        : "Light Sensor",
    "deviceType"    : "Aeon Labs UPnP Light Sensor",
    "attribute"     : {
        "name"          : "State",
        "parameters"    : [
            {
                "name"      : "State",
                "value"     : 69,
                "dataType"  : "time",
            }
            ...
        ]
    }
}
```

3.4 Development

3.4.1 Deployment

Introduction

This section outlines the details about building and deploying the Aria system. It details how the system's automated build process executes and the build manifest. This document also outlines the major decisions that were made to run the build and the reasoning behind them.

Build Process

The build process uses a centralized manifest structure to execute the build. The build system does not use a build automation suite but instead integrates a number of different technologies for deployment. The executing process that runs the build is a simple shell command executor that runs over a set of commands from a central manifest.

The central manifest contains all of the build commands for each sub-project. It is a collection of bash commands that are executed in order to build the system. This manifest file is located under the build directory and is named `manifest.json`.

The Aria system is composed of many subsystems that each require their own build tool chain. Each subcomponent has its own set of build commands which are encapsulated within the central manifest. For this reason, it was decided that a simple command executor would be more suitable for running the main build process rather then having another architecture to try to execute the commands.

The deployment of the Aria system is not restricted to a single platform but is instead intended to operate on many different operating systems with many different package managers. To accommodate this requirement, a package manager wrapper was built. This wrapper dispatches commands to the appropriate system package manager to download any required dependencies for this project. This is used in conjunction with the main build system, allowing the Aria system to be deployable on numerous architectures.

Package Management

Almost every system has a different set of package management tools. Most of these tools support the same features and can access the same packages for their specific system. To enable the Aria system to build on independent systems, a package manager wrapper named `pkman` was developed. This wrapper offers a standard set of package manager commands to install and uninstall packages using the target platform's specific package manager. This tool uses the facade design technique and allows the build process to use one generic interface to communicate to many different dependency management tools.

Since not all platforms offer the same set of packages, a special mode was added to `pkman` to support optional dependencies. Optional dependencies can be added by specifying a `--try` flag which indicates that if the package install does not succeed then continue without error. This is a major improvement that allows us to support packages which have different names per platform or even some that only exist on specific platforms.

Building Aria

The Aria system is built in several stages. These stages are executed in order to setup the target system's architecture for running Aria. Each stage is referred to as a directive in the build process. To perform a complete build these directives must be executed in order and then must be deployed.

Executing a Build

To execute a build in the aria system simple run the `./build/run all` command. This will execute all of the stages of the Aria build process including the Aria test suite. Once the build has been successfully built it can be deployed with the `./build/run deploy` command. This will start the Aria system processes in a daemonized state.

To execute a specific directive in the build, simply run the `./build/run` command with one of the following directives.

Directive	Description
`enviro`	Setup the target system's environment dependencies
`deps`	Install all of the target specific technologies
`build`	Build all of the targets on the build system
`test`	Execute the automated test suite on the build
`deploy`	Start running the Aria system
`clean`	Clean all of the build artifacts from the repository
`all`	Install all of the dependencies, build the system, and run the tests

3.4.2 Technologies

The Aria system is composed of many different technologies that each serve a task specific purpose. This section outlines the details about the technologies chosen and the reasons for them being selected. This section is broken down into the different subsystems of the Aria system.

Exchange Server

The exchange server is responsible for logging and sending messages throughout the Aria system. It functions as the central communication point for all messages within the Aria system. This server needs to support the main communication protocols that are required for interfacing with several different protocols, including UDP, UPnP, Z-Wave and others. To accommodate this requirement, Python 3 was selected as the primary language.

Python 3 offers support for most of the desired protocols for this system. It can directly interface with C allowing for support of any external library implementations. As for the version decision of Python, choosing version 3 over version 2 allows us to use many of the new features that Python offers. It also means that we are not writing Python using a set of deprecated standards.

Database

The central exchange server needs to record all events and messages that it receives. This database will be accessed by the exchange server and could be integrated into the exchange process. The database needs to have transaction management but does not need to be a sophisticated database server. SQLite provides all of the required features for this operation and provides the system with a relational data store. Using SQLite means that the system will need SQL to read and write data from the database.

Quality Control

To maintain the desired level of quality in development, tooling is needed to validate the system. Both static and dynamic techniques for quality control are being used to aid in the development of the smart home system. For static code analysis of the exchange hub, **pyflakes** is being used to stop syntax errors. This tool reads a python file and reports if it conforms to a set of desired constraints.

To check the system's dynamic functionality, the system has sets of integrated unit tests. Python comes with a built in unit testing framework appropriately named **unittest**. This package is being used to develop test stubs and test cases to validate the operations of the exchange. To drive these tests, **nosetools** is being used. This tool automatically discovers test cases and manages running the test cases in a contained environment. The two tools complement each other and create the basis for the exchange unit testing structure.

Communication Libraries

In order to interface with different communication protocols, the central exchange is required to use a number of different third party libraries to communicate. These libraries provide direct translation from the exchange's internal message structure to the target devices communication protocol.

WeMo

The first of the these communication libraries is used to interface to WeMo devices and is titled **netdisco**. This library provides the exchange command with the ability to discover WeMo devices, get device states, as well as send control commands.

HTTP Gateway

The HTTP Gateway is responsible for serving static web content as well as enabling communication from the web client to the exchange server. The gateway creates a thin wrapper around the internal communication structure of the exchange server and serves it to the web client with a RESTful API. To perform these tasks a simple new web technology, **node.js**, was used. Node is a JavaScript interpreter that provides a massive set of server development tools and libraries.

One of the easiest to use node packages is a web server called **Express**. Express is dynamically configured by coding to its interfaces, which enables rapid development of a web server. This technology has been used for the HTTP gateway to simplify the development.

Deployment

In order to compile the server into a single executable, **Webpack** is being used. Webpack bundles all of a systems source code into a single file and adds a node.js 'shebang' (`#!/usr/local/env node`) to the start. This process allows the simple JavaScript files to be turned into an executable bundle.

Package Management

With almost any node.js project there are dependencies. This project is no exception as it uses several packages for runtime, deployment and testing. Fortunately, node.js comes with a default package manager appropriately named **Node Package Manager** (NPM). NPM is the standard for all node.js projects and is consistent across various operating systems.

Quality Control

To ensure that the code for the hub gateway is adequate and dependable, multiple testing and validation techniques are used. First, the gateway code goes through a static analysis tool called **JSHint** that checks for syntax and lexical errors. This phase quickly indicates where issues may lie in code before it is even tested.

Static testing is quick and often useful, but does not test the execution of the gateway. In order to dynamically test this gateway, the **Mocha** unit testing framework was added. This framework provides a behaviour driven development (BDD) testing language for creating unit tests for the gateway.

Web Client

The web client is the front end facing user interface that controls the Aria system. The client is responsible for providing observability of the system as well as controllability of the hub and various devices. The web client is intended to be consumed in a user's web browser and therefore must use the language of the web, JavaScript.

Deployment

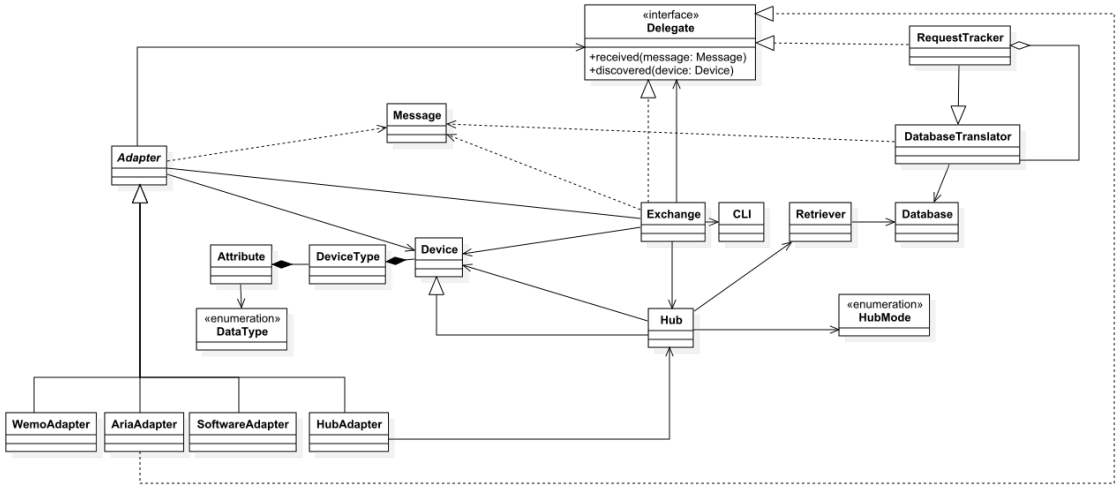
Deploying code to different web browsers tends to be a bit tricky, as most browsers deviate on their implementation of the JavaScript language. To ensure that the web client will operate on the lowest common denominator of browsers, a translation layer was used when building the client. This translation tool, called **Babel**, allows JavaScript to be written using the newest version of the EMCAScript specification, `es6` . This allows the JavaScript to still function on browsers that only support the older version `es5` . It does this by transpiling all of the new JavaScript features into the old version equivalent ones.

Once the web client has been converted into browser executable code, it is then bundled into a single web app file using **Webpack**. This is the same tool that is used for the gateway to make it executable. Here the intent is to obfuscate and minimize the size of the web application so that it has a faster load time on client.

Quality Control

Similarly to the gateway, the web client undergoes two phases of testing; static and dynamic. The static phase is exactly the same as the gateway as it is analyzed by JSHint. The dynamic phase slightly differs as the target platform is vastly different from the gateway. In order to dynamically tets the remote, a testing library called **Jasmine** was used to write unit tests. This library is similar to Mocha in that it provides a BDD testing grammar for creating unit tests. To run these tests, a test driver called **Karma** was used in conjunction with a daemonized browser called **PhantomJS**. The Karma driver runs the Jasmine tests in the PhantomJS browser to simulate running tests in an end user's browser.

3.4.3 Hub Implementation



Adapters

The adapter classes are used to discover devices and communicate with them. The adapters use an observer observable model to notify delegates when they receive a message or discover a device. Each adapter is responsible for creating a Device object that can be controlled by the system for each new device they discover.

Aria Adapter

The Aria Adapter is used to communicate between the Exchange and the gateway as well as it can communicate with any custom device we create. It uses a UDP socket that listens on port 7600. The Aria Adapter is also a delegate on the Exchange Object and is able to push events and newly discovered devices to the gateway.

Hub Adapter

The Hub adapter is used to send and receive messages for the hub. The adapter translates the message and calls the appropriate method on the Hub.

WeMo Adapter

The WeMo adpater is responsable for communicating with the WeMo UPnP devices. The adapter is able to discover new WeMo devices that are added to the system as well as create a python object from the SOAP xml that the device provides to allow the system an easy way of controlling the device.

While trying to integrate with a WeMo switch we first looked at the python `ouimeaux` library. This library looked very promising as it had discovery and created python objects for each device that would notify using `pysignals` if the state of the device changed. The issue with this library is that at the time of writing this report there is a bug with processing signals that exists when trying to run with python version 3.4 or higher. As we are using python 3.5 for our hub this meant that this library was therefore incompatible.

We then looked at the `netdisco` library that provides discovery for UPnP devices. Using this library we are able to discover our WeMo devices but we didn't have any objects that we could use to control the devices. In order to create objects from the discovered devices we used the `pywemo` library. This library also allowed us to register to events when the devices' state change so we get notified and are able to log an event.

Database

The Database class is used to execute SQL on the sqlite database. The Database will create a new database if one doesn't currently exist when the Hub is started.

DatabaseTranslator

The DatabaseTranslator translates event and request messages into sql and sends it to the database to be stored.

RequestTracker

The RequestTracker listens to the Exchange and decorates the DatabaseTranslator, filtering out all requests that have the Hub as the destination. These can be ignored because these request are for getting information about the system and don't need to be stored for machine learning. The RequestTracker also associates a request with an event so that any changes the request made can be tracked. If a device that can be controlled sends an event and there is there was no request to that device a request is created by the request tracker in order for the user action to be tracked.

Retriever

The Retriever class is used to query the database and retrieve events. Currently it is possible to query for a list of all events or a list of all events for a specified device.

Delegate

The delegate interface is an Observer interface. The interface implements two methods; received and discovered.

Device

A device is a representation of a device that is connected to the system and can provide input or be controlled.

DeviceType

The Device type represents the different types of devices that are in the system. The device type contains information about the manufacturer of the device the protocol the device speaks and the different attributes that the device has.

Attribute

Attributes represent the different attributes of a device that can be viewed and/or changed

DataType

Data type is an enum that is used to represent the data types of device attributes. The possible values of this enum are:

- 'binary'
- 'int'
- 'float'
- 'colour'
- 'enum'
- 'time'
- 'date'
- 'string'
- 'list'

CLI

The CLI provides a command line interface for the user to issue command directly to the hub. Current commands are:

- help
- exit

Hub

The Hub is a device that contains the current system state. The Hub has a list of all of the devices in the system, what mode the system is currently in and is able to query the database to get events.

Exchange

The Exchange is used to route messages through the system. The exchange listens to all of the adapters and receives all the messages that are passed through the system.

HubMode

HubMode is an enum that contains the fdifferent modes of operation of the hub. The following are the possible values:

- StandBy
- Normal
- Learning

3.4.4 Communication with Z-Wave Devices

The system uses the Aeotec Z-Stick Series 2 from Aeon Labs to communicate with Z-Wave devices from the system's hub. The Z-Stick is attached to a USB port in the system hub device (Raspberry Pi).

Controlling the Z-Stick

When connected to a USB port, the Z-Stick exposes a proprietary serial API, which can be used by controller software to manage a network of Z-Wave devices. This document describes how the hub software uses the Z-Stick to communicate with devices in the home's network.

Z-Stick API

There are several challenges to overcome in order to write Z-Wave controller software that uses the Z-Stick.

- Proprietary API: The serial API exposed by the Z-Stick is not made public
- Low-level: The Z-Stick API is very low-level; significant effort is required in order to use the most basic features of Z-Wave devices

In order to avoid the difficulties of using the Z-Stick API directly, we chose to search for a Z-Wave controller library which provides a higher level of abstraction away from the Z-Stick API.

Library Alternatives

Comparison Criteria

- Does the library have a Python binding? The hub code is primarily written in Python, it is less effort to include a Python library as opposed to a C/C++ library
- Is the library mature enough to be a reliable part of the project? If many other projects use the library already, with positive results, this indicates that the library is stable enough to include in the project
- Is documentation and tutorial material readily accessible?

Sigma Designs Developer Kit (ZWare)

Sigma Designs is the primary company involved in development of the Z-Wave protocol. Sigma Designs provides an developer kit, targeted at companies that are attempting to build custom Z-Wave devices. The targeted applications for the library are listed as: door locks, lights, sensors, and thermostats.

For commercial applications, the developer kit also includes licenses which allow custom Z-Wave devices to be sold legally [66].

The developer kit provides access to the following software resources:

- Documentation for the Serial API
- Z-Ware C API
- Sample applications

OpenZWave

OpenZWave is a free open-source library for communicating with Z-Wave devices using compatible ZWave controller devices. The Aeotec Z-Stick is fully compatible with OpenZWave. OpenZWave is a C++ library; the developers have created bindings for several other languages including Python.

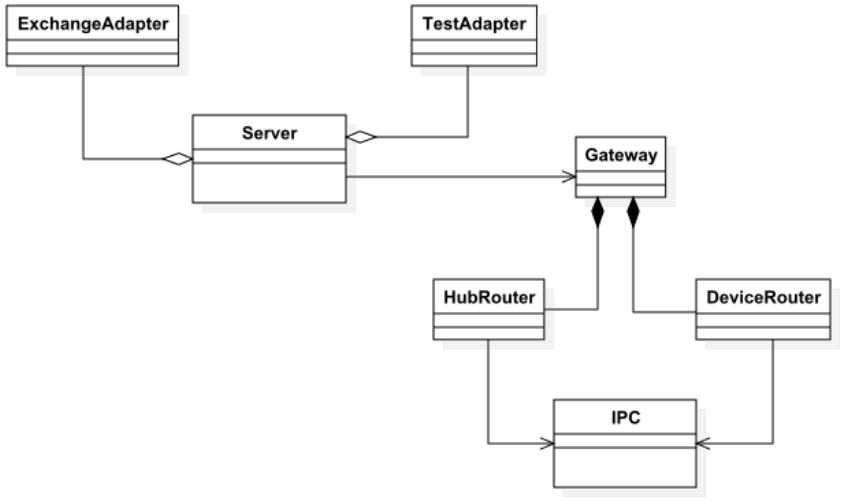
The OpenZWave source code is well-documented, and includes several examples of usage. OpenZWave is used by several existing home automation products to communicate with devices, with 259 forks on Github and an active community of developers [67]. Additionally, OpenZWave is a member of the Z-Wave Alliance, the consortium of companies that certifies Z-Wave devices [68].

Comparison

Library	Python Binding	Maturity	Availability
Sigma Designs	No binding	Official SDK	License Required
OpenZWave	Has a binding	Mature	Open Source

We decided to include the OpenZWave library in the project. The availability of a Python binding for the library makes including it in the project simpler than the Sigma Designs C API. The Sigma Designs Developer Kit is also targeted at companies that are attempting to create their own certified Z-Wave devices, which is outside the scope of this project. Tutorials and examples of using the OpenZWave library are also more accessible online, from a variety of sources.

3.4.5 Gateway Implementation



Exchange Adapter

The exchange adapter is responsible for communicating with the exchange server using inter process communication with a connectionless socket. When the exchange adapter registers itself with the exchange server it provides a callback port for asynchronous callback events. This allows the exchange server to push communication to the gateway, which in turn forwards them on to the remote client.

Test Adapter

The test adapter allows the server to run independently of the exchange server and rely only on integration tests. The integration tests provide a complete mock of the IPC protocol so that all exchange communication is controlled. This is used for testing the gateway and remote client in isolation from the exchange server.

Server

The server class is the main class for the gateway project. This class initializes the appropriate adapter object for the gateway class and binds itself to a port for serving data.

Gateway

The gateway class provides a REST interface for communicating to the system over HTTP. The gateway delegates the endpoints of the REST API to various routers. Internally, this is done using the strategy pattern. As a request enters the system, it is routed to the appropriate handler based on its URL.

HubRouter

The hub router handles all requests for the hub endpoint. This router deals with communication that will request or update the state of the central hub. The hub router is also responsible for retrieving event logs from the hub's database.

DeviceRouter

The device router handles all individual device requests. It is responsible for updating individual device states as well as querying all device data.

IPC

The IPC class is responsible for communicating to the exchange server. It serializes and deserializes messages using the defined IPC protocol (see IPC).

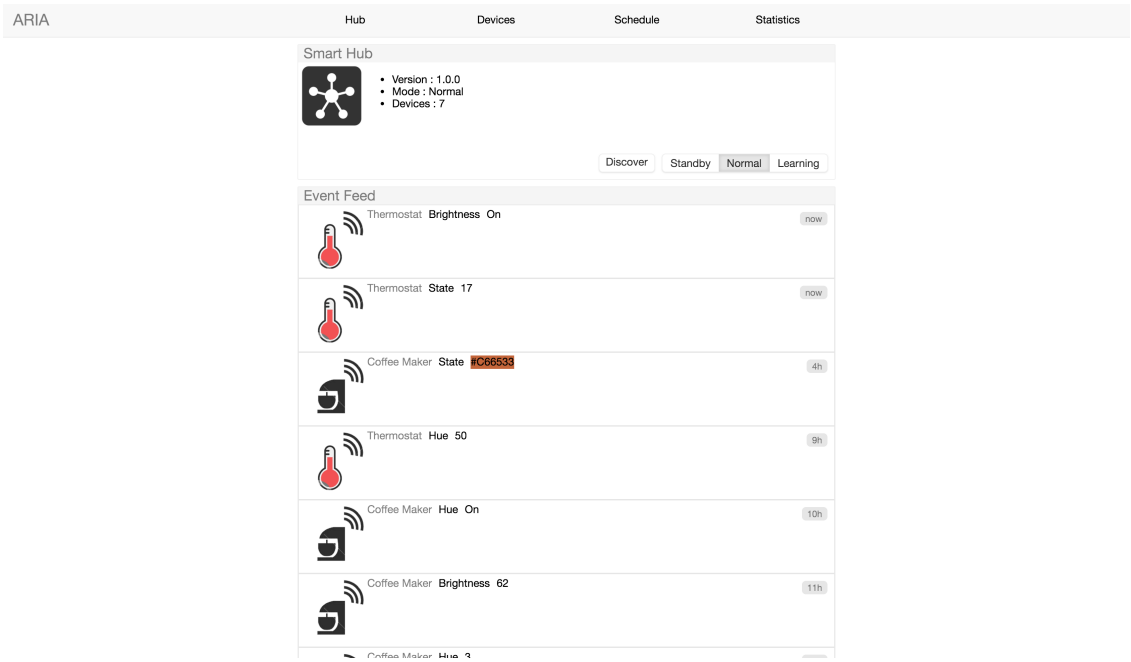
3.4.6 Remote Implementation

The remote client provides a user interface for the end user to control and observe the state of the system. It is responsible for providing events and notifications to the user about the current operations of the hub and its devices. The remote must also provide

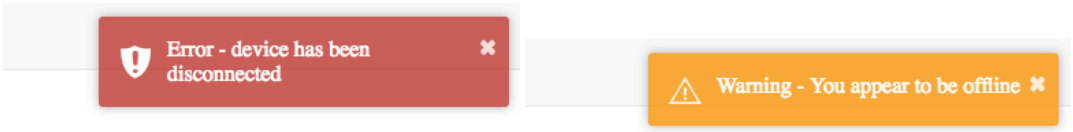
controls for changing the state of all components within the system.

User Interface

Hub Page

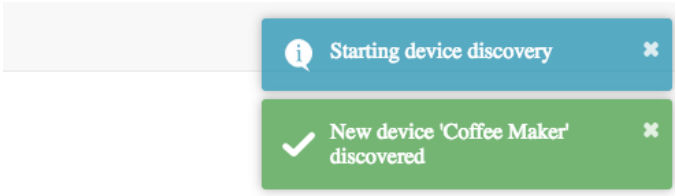


This page provides the user with controls for updating the state of the hub and discovering devices. In the lower area of the page there is an event feed that is updated using push notifications whenever an action happens within the system. This provides the user with details about the behaviours in the system, what is changing and when it happened. To notify the user when errors or warning happen within the hub, push notifications are used.

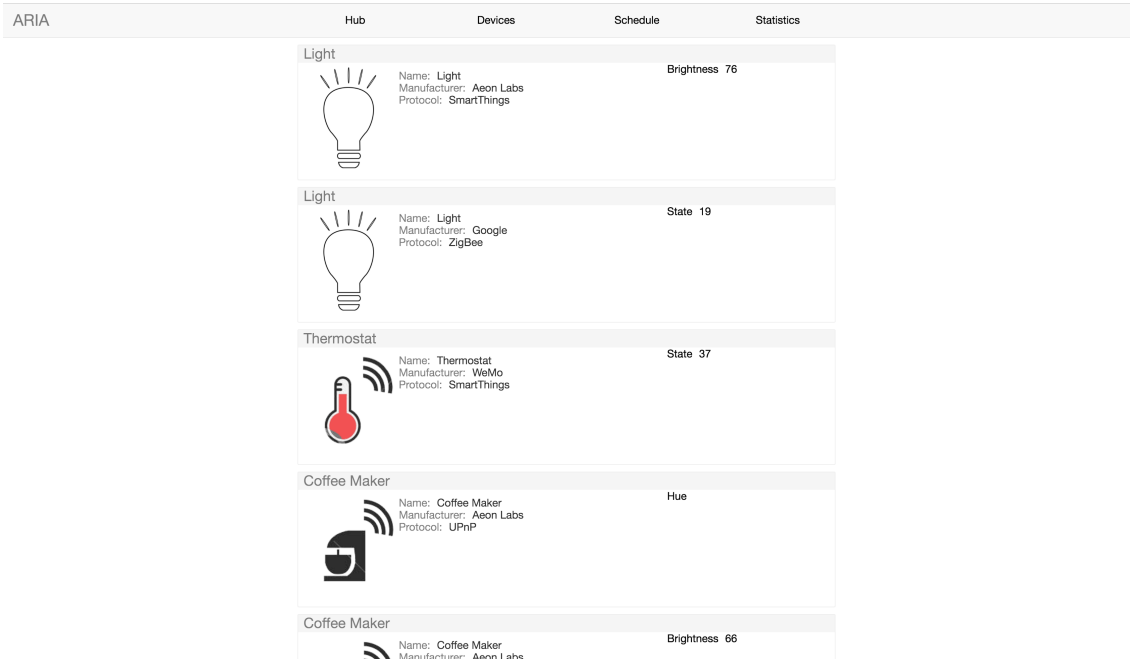


Discovery

When the user presses the discover button, the system begins a device discovery sequence. It begins searching for new devices in the local network area. If a device is found by the hub then it is added to the hub. The user is notified of the addition of these devices through push events as seen below.



Device Page



This page is responsible for providing the user with state information and controls for the devices within the system. Each device's current state is displayed to the user in the device list. Currently, there are no controls for a device but they will be implemented using the wire frame as discussed in the remote user interface design

Implementation

The remote client is designed entirely using HTML, JavaScript and CSS. For more information about the technologies used, see web client technology.

All of the client code follows a *reactive* pattern where all views update as state changes. All view components in the system follow the composition pattern and inherit from a common component class. Whenever state changes, each component follows a transition pattern to redraw itself with the new state. First, the state of the component is *updated* which in turn updates all of its child components. Next, the interface is re-rendered. This is done by calling *render* on the top level component that has changed. When a component is rendered it is responsible for calling *render* on all of its children creating a trickle effect. Through this method, calling *render* on the top level component will re-render all of the changed elements.

3.4.7 Lighting Scenario Data

Purpose

The purpose of this section is to present analysis of actual device data for the Smart Lighting Scenario.

Devices

- Aeon Labs Z-wave Multisensor 6
- LB60Z-1 Z-wave Dimmable lightbulb

Steps to Produce

1. Motion sensor and Dimmable LED set up facing each other in empty room.
2. User enters the room
3. User turns LED on to 100% brightness
4. User decreases LED brightness to low value
5. User blocks multisensor with hand
6. User increases brightness level to 100%
7. User turns off LED from Web UI and leaves the room

Data

```
SELECT event_id,
       e.timestamp,
       e.source,
       p.NAME,
       pc.value,
       e.request_id,
       r.timestamp AS request_time,
       r.source,
       r.receiver
FROM   parameter_change pc
       JOIN event e
         ON e.id = pc.event_id
       JOIN parameter p
         ON p.id = pc.parameter
       LEFT JOIN request r
              ON e.request_id = r.id
```

event_id	timestamp	name	value	request_id	request_time
5	2017-02-01 03:57:12	Luminance	0.0		
10	2017-02-01 03:57:19	Luminance	0.0		
15	2017-02-01 03:57:25	Luminance	0.0		
17	2017-02-01 03:57:26	Motion	1		
18	2017-02-01 03:57:27	Level	99	1	2017-02-01 03:57:26
19	2017-02-01 03:57:27	Alarm Type	0		
20	2017-02-01 03:57:27	Alarm Level	0		
22	2017-02-01 03:57:27	Burglar	8		
26	2017-02-01 03:57:32	Luminance	4.0		
31	2017-02-01 03:57:38	Luminance	4.0		
36	2017-02-01 03:57:45	Luminance	4.0		
38	2017-02-01 03:57:49	Level	36	2	2017-02-01 03:57:49
42	2017-02-01 03:57:51	Luminance	1.0		
43	2017-02-01 03:57:51	Level	98	3	2017-02-01 03:57:51
48	2017-02-01 03:57:58	Luminance	4.0		
53	2017-02-01 03:58:04	Luminance	4.0		
55	2017-02-01 03:58:09	Level	0	4	2017-02-01 03:58:09
59	2017-02-01 03:58:11	Luminance	4.0		
64	2017-02-01 03:58:17	Luminance	0.0		
69	2017-02-01 03:58:24	Luminance	0.0		
74	2017-02-01 03:58:30	Luminance	0.0		
79	2017-02-01 03:58:37	Luminance	0.0		
84	2017-02-01 03:58:44	Luminance	0.0		
89	2017-02-01 03:58:50	Luminance	0.0		
94	2017-02-01 03:58:57	Luminance	0.0		

3.5 Testing

This section describes how the system's components are tested. The section is organized by component; for a description of the purpose of each component, please see System Components.

3.5.1 Hub Testing

Context

The DeviceCommunication, CommunicationServer, and EventLogger components form the core of the system's hub. Each of these components is written in Python, and work together to react to events from devices. They are also responsible for fulfilling requests from the HTTP Server.

Events from connected devices are first received by the DeviceCommunication component, which forwards messages to the CommunicationServer component. CommunicationServer routes messages to the appropriate component, which may be another device HTTPGateway or EventLogger. This section describes the unit testing of each component, as well the method used to test that the components perform correctly together.

Unit Testing

The modules that make up each component are unit tested using Python's *unittest* framework. The unit tests are used for regression testing, and are written by the same person who implemented the unit under test.

The *unittest* framework makes use of Python's `decorator` construct to provide an easy way to isolate components for testing. Dependencies such as threading, database, and network libraries can be replaced with a mock obejcts automatically by declaring a test case with the `@unittest.patch(<dependency-name>)` decorator. These mock objects are used as stubs to provide control the behaviour of the dependencies of a module. For example, mock objects are used to return test data when a module attempts to read data from a network socket.

Integration Testing

Interactions between each of the components are triggered by the receipt of messages from smart devices, as well as requests to the REST API. In order to make tests repeatable and to minimize testing time, we chose to test the integration of these components using mock device data. Using mock device test data also allows development of the components to proceed before the implementation of communication with physical devices is complete.

The DeviceCommunication component makes use of an adapter pattern to provide communication with different devices (which may have varying interfaces), through a common interface. This structure provides a simple way to simulate events for testing purposes; an adapter was created which generates events from software source. Automated tests enqueue various messages in the test adapter, which are then propagated through the system as if they came from a physical device.

The choice of SQLite as a database engine makes the system easier to test. SQLite databases are contained in a single file; this makes it inexpensive to tear down and re-create a database which is used for testing. The simplicity of SQLite allows each test case to write to its own instance of the database. The advantage of having one database per test case is that any test failures can be debugged in isolation because the state of the database after a test faiure is preserved.

The suite of integration tests also makes use of python's *unittest* framework to create test drivers. Unlike unit tests, the integration test suite does not mock out dependencies such as threads, database, etc. The *unittest* framework is used only to automate the execution of tests, so the behaviour of the components is as close as possible to their actual behaviour when the entire system is running.

3.5.2 Remote Testing

Context

The gateway and remote client work closely to provide control and observability to the Aria system. These components are written in JavaScript and need to communicate using a REST API. The gateway is responsible for translating REST communications to the required IPC communicates for all operations. The remote is simply a user interface to interacting with the controls provided by the gateway.

Unit Testing

The gateway is executed in node.js, a server side environment for JavaScript applications. In order to test the internal behaviours of the gateway, a common BDD testing framework called *mocha* was used. Mocha allows for simple feature based unit tests to be constructed and executed in the node.js environment. Mocha also provides capabilities for spying and mocking internal classes from the gateway code.

The remote client, although written in the same language, runs in very different environments. The web client has to be able to run in a browser. In order to test the functionality of the remote, a unit testing framework called *karma* was used. Karma is a test runner that uses *PhantomJS* to execute its unit tests in a browser environment.

This unit testing library is run against every build of the system and is used for regression testing purposes.

Integration Testing

In order to isolate the behaviours of the gateway and remote from the exchange server, an interactive integration testing suite was added. This testing suite mocks all communication between the gateway and exchange server in order to control the behaviour of the system. This integration test allows the gateway to run, server the remote client and execute all of the behaviours described by its API. This form of testing has been used to manually test the user interface functionality in a controlled environment.

To start the gateway's integration tests, the gateway executable simply needs to be passed the `--test` flag. This will launch the gateway with all normal parameters but will swap the exchange communication adapter with a mocked adapter.

3.6 Project Progress

This section outlines the details about the implementation of the Aria system. It is a record of the major decisions made in the implementation of the system as well as a record of the progress of the development. The details of the project includes the original proposed timeline for the development of the system with a comparison to the true timeline.

3.6.1 Proposed Timeline

At the beginning of the project, the development of the project was planned in a week to week plan. This development plan had fairly restrictive timelines and did not leave much room for error. The table below is the originally proposed timeline.

Milestone	Date
Arduino micro-controller hooked up to voltage relay	October 5, 2016
Central server communication with light sensor values	October 12, 2016
Light sensor communication to light controller	October 19, 2016
Arduino motor that moves curtain track	October 26, 2016
Add communication to curtain Arduino	November 2, 2016
Web API	November 9, 2016
Static functioning web interface	November 16, 2016
Add temperature control buttons	November 23, 2016
Add temperature device communication	November 30, 2016
Add basic decision making software	December 7, 2016
Add device discovery	January 2, 2017
Improved web client	January 11, 2017
Improved decision making	January 18, 2017
Add record and learn	January 25, 2017

It quickly became evident that this timeline was too rigid and was not representative of the true development of the project. At the beginning of October, the vision of the project was more formally defined with a much clearer end goal for the project. After the project had been more formally defined, the original timeline was no longer representative of the goals set for the project.

In order to properly plan the development with these new goals in mind, the timeline had to be revised. The new timeline for

the project does not rely on dates for individual tasks but instead has set milestones with dates and the tasks that need to be completed for that milestone. Below is the revised timeline for the project. The first milestone includes the requirements elicitation which was formalization of the project goals.

Milestone	**Date**
Requirements Elicitation	October 12, 2016
Discovery of devices	
Basic control of devices	
Device displayed in user interface	
Event recording in database	
Events displayed in user interface	
Device control from the user interface	
Basic decision system	
User interface control for basic decisions	
Discovery of devices from user interface	
Oral Presentation Demonstration	January 27, 2017
Automated machine learning model builder	
Decision making software to apply model	
Light scenario complete	
Temperature scenario complete	
Coffee scenario complete	
Poster Fair	March 17, 2017

This new timeline defines three major milestones that represent concrete deliverable times for the project. The intermediate goals are all required to be completed before their respective milestone. To ensure that the project is remaining on schedule, this timeline is reviewed on a weekly basis and used to schedule and prioritize project tasks.

3.6.2 Project Status

The project planning and development has been an intertwined process that has resulted in a more focused direction of the project on a week to week basis. Every week the team selects a set of tasks to focus on in order to achieve the next major milestone. This process has been iterative and is allowing for incremental development of the system.

Current Status

The next major milestone for the development of the system is the **Oral Presentation Demonstration** in January. In order to reach this milestone a number of basic system features need to be in place and operational. The details of these items are outlined in the revised proposed timeline table above.

The system currently has a large portion of these items working. More specifically, the system can currently perform the following tasks:

Discovery of devices

The user is able to add devices to the system by connecting them to the network and following the discovery sequence. Discovery is different for different devices and protocols. The system is able to automatically discover Z-Wave devices using the OpenZWave library.

Device displayed in user interface

All devices that are registered in the system are visible from the user interface. The remote web client provides observability for the specific devices in the system as well as their currently reported status. Some basic metadata including name, manufacturer and protocol is also displayed to the user.

Events displayed in user interface

The user interface currently uses push notifications to receive events as they are logged in the system. These events are then immediately displayed to the user in a live updating event feed.

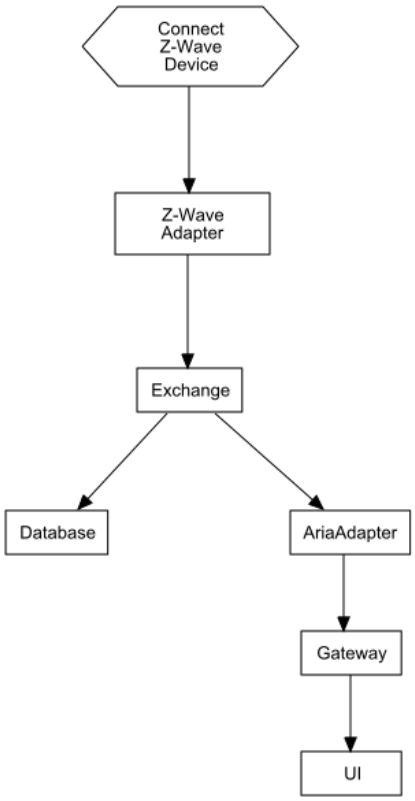
Discovery of devices from user interface

The user interface provides a mechanism for launching an automated discovery probe. This process kicks off discovery for any devices to be added to the system. If any new devices are added to the system during this process, the user is notified and the user interface is updated.

3.7 Data Flows

3.7.1 Add Device Data Flow

This data flow outlines the details of how a new device is connected to the system. In particular, this workflow focuses on adding a Z-Wave device to the system after. This workflow assumes that the system is already online and operational, that the Z-Wave stick is plugged into the smart hub and that the network is live



When the Z-Wave adapter starts up, it uses the OpenZWave library to initiate a Z-Wave network. When a Z-Wave device is first paired with the Z-Stick (which is attached to the Raspberry Pi), the network initiated by our Z-Wave adapter receives a device object. This device object has been built by the OpenZWave library. An example of a such an object is shown below. For all of the data received from the device see Appendix-G

```
basic:4
product_name:LB60Z-1 Dimmable LED Light Bulb
device_type:Light Dimmer Switch
values:{
```

```
   72057594093076481:{
      'label':'Level',
      'data':99,
      'genre':'User',
      'value_id':72057594093076481,
      'units':'',
      'node_id':3
   },
   72057594093076504:{
      'label':'Bright',
      'data':False,
      'genre':'User',
      'value_id':72057594093076504,
      'units':'',
      'node_id':3
   },
   72057594093076520:{
      'label':'Dim',
      'data':False,
      'genre':'User',
      'value_id':72057594093076520,
      'units':'',
      'node_id':3
   },
}
node_id:3
location:
name:
product_name:LB60Z-1 Dimmable LED Light Bulb
type:Light Dimmer Switch
manufacturer_name:Linear
name:
home_id:4036346586
```

Once this ZWave object is received by the adapter, it is converted into a generic device object which can be used internally in our system, shown below. We also set the location on the ZWave device to a UUID we have generated so that we are able to track this device if it is ever disconnected from the system.

```
name: LB60Z-1 Dimmable LED Light Bulb
DeviceType: {
                LB60Z-1 Dimmable LED Light Bulb
                protocol: zwave,
                maker: Linear
                Attributes:
                [
                   {
                       name: Bright,
                       parameters:  [
                                        {
                                            name: Bright,
                                            DataType: binary,
                                            value: False,
                                            min 0,
                                            max: 0,
                                            step: None
                                         }

                                    ]
                   },
                   {
                       name: Level
                       parameters:  [
                                        {
                                            name: Level,
                                            DataType: byte,
                                            value: 99,
                                            min 0,
                                            max: 99,
                                            step: None
                                        }

                                    ]
```

```
                    },
                    {
                         name: Dim,
                         parameters: [
                                        {
                                            name: Dim,
                                            DataType: binary,
                                            value: False,
                                            min 0,
                                            max: 0,
                                            step: None
                                        }

                                       ]
                    }
                  ]
            }
address: abb757fc-5d38-45c8-bbbc-81cf6bf29d92
version: 4
```

Our ZWave adapter now notifies the Exchange that a device has been discovered, passing it the new device object. From here, the device must be stored in the database, and the UI needs to be updated to display the new device and its information. The exchange forwards the device object to the database and to another adapter, which formats it properly for the UI.

The device adapter is responsible for doing this formatting. In order to be properly displayed on the UI, the device object must be converted into a JSON string. Once this is done, a device discovered message is sent to the Gateway.

The purpose of the Gateway is to take messages received over a UDP socket and forward them over a Web socket to the user interface.

4 Conclusion and Recommendations

5 Appendix

A Automation Systems

Background

This section contains research on existing home automation systems in order to better understand what technologies currently exist. This research is intended to help understand how a new system could improve on existing technology or if there are particular architecture patterns that are relevant for this system's design. In particular, this research focuses on features that are common between home automation systems and their benefits to the end user.

Since usability is the focus of this system, it is important to investigate the features that a user would expect in a smart home system. It is also important to examine any short comings of existing systems and if these short comings can be addressed.

In order to compare these smart home systems, a set of criteria is required such that they can be evaluated. For this research, the following criteria will be examined: user interaction, device discovery and setup, network configuration, communication protocols, application programming interfaces, third party integrations, and limitations. If there are common protocols across all main automation systems then they should be considered for this system. Industry standard protocols may be discovered by reviewing these existing systems. If there is a special focus on the ease of use then this system should consider the efforts that other systems have taken. The network configuration of a system will govern its performance, power consumption and allowable number of connections. This research will investigate how these systems organize their network topologies to satisfy these quality attributes. In order to control these devices we require that the following; a way to view all connected devices, a way to get notifications when a device in the system changes states, a way to change the state of a device. Finally, if there are accepted standards of third party integrations then this system should consider them in its design to maximize overall interoperability.

A-1 Insteon

Description

Insteon is a home automation system that allows smart devices to connect over a home area network with a focus on ease of use. Insteon allows you to monitor and control all of the devices within the system. All devices in connect through a central Insteon smart hub which is then controlled by the user through a app on the user's mobile device. Controlling devices can also be automated using manually configured schedules, IFTTT or through the use of **scenes**.

A scene allows a user to configure device behaviours based on grouping of devices in their home. The devices are grouped into rooms which can represent the physical rooms of the home. The user can then create a configuration for a specific date, time or environmental condition. Insteon then monitors the state of the home for these conditions, when they are met then it recreates the user's configuration [3].

Technical Overview

User Interaction

In order to communicate with devices, Insteon provides a central hub that acts as the main point of control for the smart home. The user will be required to pair any devices that are added to the smart home with the hub so that it can be controlled [4]. Once the devices have been paired with the hub, the user can control the devices remotely using the Ineston smart phone app. The Insteon app gives the user the ability to monitor and control all of the devices in their smart home, automate tasks with schedules, build scene configurations, and receive alerts about devices in their home [5].

Device Discovery and Setup

Initial setup of the smart home requires the Insteon hub to be installed in the home and connected to a network [6]. Once the hub has been connected to a network it must be paired with the user's smart phone app so that the user can access the Insteon user interface.

Adding a new device is done through the Insteon app. A new device is first plugged in and turned on. The device then locates the user's mobile app on the network and is ready to be connected. Using the app the user selects the new device and enters the device identifier. Once the identifier is entered the device is able to be configured and added to the Insteon network [7].

Network Configuration

Insteon uses a central hub to communicate between the devices and the users app [8]. The Insteon hub is used for all device communications and control. Devices connect and send all event information to the central hub for processing and storage. The user can then interact with the devices in their network through the Insteon smart phone app.

Insteon uses a peer-to-peer network to connect the devices [9]. All of Insteon's devices can act as a controller to send messages, a repeater to forward messages or a responder to receive messages.

Supported Communication Protocols

Insteon uses a proprietary messaging protocol to send messages between devices and a smart hub. The Insteon messaging protocol uses message repeating to send messages across a peer-to-peer network of devices. This messaging protocol is outlined in a detailed white paper that describes the network protocol as well as the messaging protocol [10].

Additionally, Insteon has introduced power lines as a medium for device communication. Instaon uses the circuits that power a home as a secondary mechanism for communicating data between devices and the smart hub [11]. This power line communication as well as the wireless Insteon protocol are explored in more details in the research of communication protocols.

Insteon also integrates with a number of third party devices and controllers. These third party devices can be added to the network and be controlled through the Insteon smart hub. The hub's devices can also be controlled through the integrations of these third party devices [12].

Application Programming Interfaces

Insteon provides a complete REST API for querying the properties of their hub. The REST API allows a client to query properties about the hub, the connected devices, any connected cameras, contacts, scenes, rooms and more. The API also provides endpoints for setting data values in devices and configuring properties of users [13].

The REST documentation does state that there are some limitations in the design of the protocol. It states that the REST API does not provide full support for configuring devices and scenes and states that the Insteon app is still required for complete access [14].

Third Party Integration

Insteon produces its own devices for most of its smart home purposes [15]. On top of these basic devices, Insteon has created integrations with a number of third party smart technology vendors. These vendors include: Nest, Logitech, Amazon Echo, Stringify, Apple HomeKit, Cortana, First Alert, Sonos, and MiLocks [16]. This exceptional list of partners demonstrates that the Insteon API is flexible enough to support many existing industry technologies. This may indicate that their platform is generic enough to support all smart device requirements. The framework developed by Insteon would be a valuable area of study for this project's design.

Limitations

As stated in the API section of this analysts, in order to properly configure all devices and scenes in the Insteon system, the Insteon app is required. This limitation impedes on the user experience of the system and would hurt the overall usability.

Summary

Insteon is a mainly proprietary home automation system that focuses on user interaction through the use of a smart phone application. The system uses a central hub to communicate to all of its devices through a proprietary communication protocol. Users are given the ability to monitor, control, schedule and configure the devices in their smart home using the Insteon app.

Insteon has a wide variety of third party integrations but offers only an incomplete API for controlling their own devices. This smart home does not allow third party code to run on their hub and requires that their app be installed on a user's smart phone in order to properly control their devices. These extensibility issues make the Insteon platform a closed, commercialized system that would not be able to support additional development of a non-partner organization.

A-2 Wink

Description

Wink is a smart home automation system that prides itself on simplicity and user experience. The Wink system provides standard smart home features such as monitoring and controlling of devices, but also offers more advanced features such as energy consumption monitoring [17]. Wink uses a central hub architecture, where all devices in the network communicate to a single "smart" hub. Additionally, Wink provides a smart phone app for users to interact with the system [18].

Technical Overview

User Interaction

User interaction in the Wink system begins with its smart hub. This hub is the central point of communication for devices in the smart home. Once devices have been added to the smart hub, the user can control the hub and all of its devices with the Wink smart phone app [19]. The app provides a clean, simple interface for monitoring and controlling all devices on the smart home network.

Additionally to the app, the Wink platform offers a custom built touch screen remote control. The remote controller allows the user to use all of the features of the smart home without the use of a smart phone app [20].

In order to offer a better user experience, Wink provides a "shortcuts" feature where a user is given the ability to control multiple devices with one command [21]. This feature allows users to setup and configure related devices to perform desired tasks. When the user invokes the shortcut, the system will apply all of the user's initial configurations [22]. The shortcut feature certainly adds value to the Wink system but does require manual configuration to setup the desired shortcut.

Device Discovery and Setup

Wink does not offer any of its own smart home devices. Instead, it utilizes the vast market of existing devices by supporting various different third party integrations. Supporting many different manufacturers means that there are two different ways for devices to connect to the Wink Hub. Pressing a button on the Wink Hub broadcasts a pairing signal across the network [23]. Any new device that receives this signal will then appear on the network, and can be viewed from the Wink app. The user then selects the new device and enters the device identifier (located on the physical device) to add it to the automation system.

If the new device requires a setup through a third party app then the new device must be added to the home network using the app provided by the manufacturer. Once it has been added through the manufacturers app, it will be visible using the Wink app. It can be added to the automation system from here using the Wink app [24].

Network Configuration

Wink's central hub provides several communication protocols, all of which are wireless. In order to communicate to the hub, the user must setup a WiFi connection. This can be done initially by connecting a smart phone to the hub's internal WiFi network and then using the smart phone app [25]. Once the hub has been setup to connect to a local area WiFi network, the smart phone app can be used to control the devices connected to the hub. If the WiFi network contains an internet link then the smart phone app can be used outside of the home to monitor and control the home [26].

Supported Communication Protocol

The Wink hub offers connections to multiple different communication protocols in order to maximize its third party integrations. As a minimum, the hub must be connected to WiFi so that the hub can connect to the Wink smart phone app. Beyond that, the Wink hub offers connections for Bluetooth, Z-Wave, ZigBee, Lutron's Caseta, and Kidde protocols [27].

Application Programming Interfaces

Wink provides a RESTful service through the Wink hub and a secondary partner PubNub. This API can be accessed through the Wink web server for third party integrations. The service offers basic smart home features including monitoring and controlling devices. Wink also offers controls for creating groups of devices that can be used for shortcuts in their service [28].

Third Party Integrations

Since Wink does not make any of their own smart devices, they rely heavily on third party integrations. The table below lists all of the organizations that have partnered with Wink to offer devices that are compatible with the Wink system.

Nest	Philips	GE	Leviton	Rheem
Honeywell	TCP	Kidde	Kwiset	Lutron
Rachio	Bali	Amazon	Andersen	Canary
Carrier	Chamberlain	Commercial Electric	Cree	Dropcam
Ecobee	Emerson	GoControl	Hampton Bay	IHome
Leaksmart	Osram			

Summary

Providing support for the most popular communication protocols allows Wink to connect with almost any device a user can purchase, making them an attractive option to consumers. Their central hub design allows them to be compatible with many third party devices as well as offer the user a simple setup experience. Finally, Wink offers a simple, elegant user interface through their smart phone app or smart remote controller. These features make Wink an appealing smart home automation system for a non-technical user.

A-3 SmartThings

Description

SmartThings is Samsung's home automation system. Similar to Wink, they provide their own app allowing a user to control devices using scheduling or IFTTT.

Technical Overview

Communication Protocols

SmartThings supports devices that communicate using the Z-Wave, ZigBee, or WiFi communication protocols

Device Discovery and Setup

Adding a new device is done through the app. A user will click "find device", prompting the hub to search for any new Z-Wave, ZigBee, or WiFi devices. When the device is found the user adds it to a room and names the device.

Network

SmartThings uses a central Hub to connect all of the smart devices. The SmartThings app talks with the SmartThings Cloud which talks to the Hub which then controls the devices.

API

SmartThings provides a Groovy API to create SmartApps that allow control of devices.

Feature	Supported
List all devices	Y
Receive update on device state change	Y
Modify device state	Y

Limitations

Requires a SmartThings hub and connection to the SmartThings cloud.

Third Party Integrations

2Gig	Aeon Labs	Amazon	Belkin	Bose
Cree	ecobee	Ecolink	EcoNet Controls	Enerwave
Everspring	Fibaro	Fidure	First Alert	FortrezZ
GE	Google	Honeywell	iHome	Keen Home
Kwikset	Leak Intelligence	Leviton	LiFi Labs	Linear
Netgear	OSO Technologies	OSRAM LIGHTIFY	Philips Hue	Remotec Technology
Samsung	Samsung SmartThings	Schlage	Sengled	Skybell
Spruce	Yale	Zen		

Summary

The Samsung home automation system provides a reasonable level of support for different communication protocols, giving it a healthy amount of third party support. This is something that we will be striving for in out project. The dependency on the connection to a cloud service is something that we would like to avoid for our project.

A-4 Apple HomeKit

Description

Apple HomeKit allows users to control their smart devices using their iPad or iPhone. Apple HomeKit does not require any central hub to control the devices, but does require there to be Apple device connected to the network at all times. If a user wants to control devices with an iPhone while not at home, the devices must be connected to an Apple product that is connected to the network, such as an iPad or Apple TV.

Technical Overview

The smart devices are able to be scheduled and controlled in groups from the app.

Communication Protocol

Apple HomeKit uses WiFi as the only communication protocol.

Device Discovery and Setup

Adding new devices is done through the app. Once a device is connected to the network it can be added to the home through Apple's app, some devices require some configuration in their manufacturers apps.

Network

HomeKit uses the homes WiFi network to connect devices and all devices on the network are able to communicate with one another.

API

Feature	Supported
List all devices	Y
Receive update on device state change	Y
Modify device state	Y

Summary

There are a few aspects of the Apple HomeKit that are not ideal for incorporation into our project. First, it is Apple exclusive, which goes against the goal of having many third party support options. The lack of a central hub also makes it difficult to having support for many different types of devices. The level of communication between devices that is offered by the Apple solution is a desirable feature, but will be difficult to achieve while maintaining diverse third party support.

A-5 Summary

A common theme among existing home automation systems is that systems with a large amount of third party support rely on a central communication hub. Having a hub allows components from different manufacturers to communicate to each other through the hub. A challenge that supporting a large number of devices presents is how to discover new devices into a system, which can be delt with in several different ways such as manual button pressing or app configuration.

Looking at these systems also presented several communication protocols to investigate, including Z-wave, Zigbee, Insteon and WiFi. Along side these are devices from many different manufacturers which operate using these communication protocols. Deciding on a communication protocol and subsequent device manufactures to support is an important part of our design for our home automation system.

B Computing Devices

Background

A computing device is any electronic device that can be used to process data and send it over a smart home network. Computing devices include computers, micro-controllers and other embedded devices. Computing devices that are being examined all offer control of external hardware interfaces. This report examines computing devices that can be used for prototyping embedded control of sensors and actuators.

Relation to System

Computing devices will be used for many aspects of the smart home system. A computing device will be used for the central smart hub as well as for various other devices in the system. Different devices will be more suitable to some tasks than others. This report compares a number of these computing options with a focus on the areas of need within the system.

B-1 Arduino Uno

Description

The Arduino Uno is a small starter microcontroller that is intended for hobby projects and newcomers to embedded programming. The Uno is intended for rapid prototyping of small circuits and small embedded systems.

Technical Overview

The Uno is a microcontroller that uses the Arduino boot loader software for launching the system and executing code. The Uno is fully compatible with the Arduino SDK that provides simple access to all pins and controls on the board. The Uno comes with a 8-bit ATmega328P processor running at 16 MHz. The system comes with 32 kB of on board flash memory as well as 2 kB of SRAM and 1 kB of EEPROM. The system boot loader occupies 0.5 kB of the flash memory capacity [29].

The Uno comes with 14 digital pins, 6 of which can be programmed to use pulse width modulation. Included in the microcontroller are 6 analog to digital input pins. The board has a operating voltage of 5 V and will accept between 7-12 V for input pins. The maximum voltage range of the input pins are 6-20 V [30].

The Uno is a medium sized microcontroller that is 68.6 mm long and 53.4 mm wide. It has a total weight of approximately 25 g. The Uno also includes an integrated USB-Mini port used for serial communication and as a power source. The board includes a 5 V DC power input for running the board continually [31].

Summary

The Arduino Uno offers fast and simple prototyping options for quickly building embedded circuits. This board is very useful for rapidly testing but would not be practical in a mass production system. For the smart learning system, this board would be optimal for experimentation and quick deployment.

B-2 Arduino 101

Description

The Arduino 101 is a more sophisticated board that uses the Intel Curie processor for computing. This board leverages the Real-Time Operating System (RTOS) developed by Intel for running the board.

Technical Overview

The Arduino 101 is the most feature rich board that Arduino is offers. It comes with a full Real-Time Operating System (RTOS) that is powered by the Intel Curie. The processor is a 32-bit, 8 MHz or 16 MHz backed by 196 kB of flash memory. The RTOS is a light weight OS that only occupies 2 kB of memory to provide managed, concurrent applications [32].

The 101 comes with a number of additional features above other microcontrollers in the Arduino family. Beyond the standard USB-Mini serial connection, the 101 offers a 6 axis gyroscope accelerometer and integrated Bluetooth. The 101 also provides a 5 V DC power input for deployment use [33].

The 101 offers many of the same prototyping capabilities as the Uno with 14 digital pins, 4 of which provide pulse with modulation. The 101 also exposes 6 analog to digital pins [34].

Summary

The 101 is a much more feature rich board than the Uno, however the added features are not applicable to this project. The 101 also has a much higher price point per unit than the Uno with no real added value. This makes the Uno a more appealing candidate for rapid prototyping and simple circuit design.

B-3 Arduino Pro

Description

The Arduino Pro is a slim, no frills version of the Ardiuno Uno. This board requires more technical knowledge than the Uno or the 101. All pins exposed on the Pro require soldering to make a connection. The Pro is intended for replication of a complex circuit design with requirements for a wide range of input and output ports.

Technical Overview

The Pro is an embedded system that leverages the Arduino boot loader and a 32-bit ATmega328 processor. The Pro has two operating speeds; 8 MHz or 16 MHz. This device has an operating voltage of 5 V but has a lower power alternative that runs at 3.3 V. The Pro has the same limited memory as the Uno with only 32 kB of flash memory, 0.5 kB of which is occupied by the boot loader [35].

The Pro offers the same pin configuration as the 101 with 14 digital pins, 4 of which can be pulse with modulation, and 6 analog pins. The Pro also provides some more advanced I/O options with a universal asynchronous receiver / transmitter (UART), serial peripheral interface (SPI) bus, and an inter-integrated circuit (I2C) connection [36].

Evaluation

The Pro has many advanced features but offers little in the way of rapid prototyping. The Pro is intended for a more advanced audience than what is required for this project and is likely not a good candidate for practical applications. However, if this system was to be replicated or redistributed, the Pro would be useful for building a final product for an unmodifiable system.

B-4 Arduino Micro

Description

The Arduino Micro is the smallest microcontroller of the Arduino family. The Micro also provides the most external ports of any Arduino making it suitable for large circuits.

Technical Overview

The Micro provides high performance embedded computing with a 8-bit, 16 MHz ATmega32U4 processor. The Micro also comes standard with the Arduino bootloader and a standard 32 kB of flash memory. The Micro has a operating voltage of 5 V and comes with a standard DC power input [37].

The Micro has 20 exposed digital pins, 7 of which can be pulse width modulation. These extra pins make the Micro very suitable for large complex circuits that require many inputs or outputs. The Micro also has 12 analog to digital pins. The Micro does lack in special features. It only offers serial communication over a standard USB-Micro port. There are other special features with this device [38].

Appropriately, the Micro is the smallest and lightest microcontroller in the Arduino family. It is only 48 mm long and 18 mm wide. The Micro also only has a weight of roughly 13 g [39].

Summary

This Arduino would be perfectly suited to a mass production environment where size and weight were valuable resources. If the learning home automation system was to be commercially produced, this could be a very valuable microcontroller. This microcontroller could be used for this system, but would take more effort for prototyping and would likely not be a suitable fit.

B-5 Comparison of Arduinos

Operation Criteria

Criteria	Arduino Uno	Arduino 101	Arduino Pro	Arduino Micro
Operating System	None	RTOS	None	None
Processor Size	8-bit	32-bit	32-bit	8-bit
Processor Family	ATmega	Intel	ATmega	ATmega
Operating Voltage	5 V	3.3 V - 5 V	3.3 V - 5 V	5 V
Input Voltage	7 - 12 V	7 - 12 V	7 - 12 V	7-12 V
Clock Speed	16 MHz	8 MHz - 16 MHz	8 MHz - 16 MHz	16 MHz
Digital Pins	14	14	14	20
Pulse with Modulation Pins	6	4	4	7
Analog Input Pins	6	6	6	12
DC Current per Pin	20 mA	20 mA	40 mA	20 mA
Flash Memory	32 kB	196 kB	32 kB	32 kB
System Size	0.5 kB	2 kB	2 kB	4 kB

Features

Feature	Arduino Uno	Arduino 101	Arduino Pro	Arduino Micro
USB	USB-Mini	USB-Mini	USB-Micro	USB-Micro
Accelerometer	No	6-Axis Gyro	No	No
Bluetooth	No	Yes	No	No
UART	No	No	Yes	No
SPI Bus	No	No	Yes	No
I2C	No	No	Yes	No

Physical Characteristics

Dimension	Arduino Uno	Arduino 101	Arduino Pro	Arduino Micro
Length	68.6 mm	68.6 mm	52.1 mm	48 mm
Width	53.4 mm	53.4 mm	53.3 mm	18 mm
Weight	25 g	34 g	N/A	13 g

B-6 Raspberry Pi Zero

Description

The Raspberry Pi Zero is a minimal computer board that offers a full computing platform in a compact form for embedded computing. The Zero is the smallest board in the Raspberry Pi family and is ideal for medium to heavy computation with minimal footprint. The Raspberry Pi Zero is one of the only Raspberry Pi boards that competes for a real embedded computing experience.

Technical Overview

The Raspberry Pi Zero has the smallest surface area of any of the Pi's, measuring in at only 65mm long by 30mm wide. To make the board as small as possible, many of the standard Raspberry Pi features were removed. This means that the Zero has no on-board WiFi, Bluetooth or even Ethernet. Despite these losses, the board is still equip with a 32-bit 1GHz Broadcom BCM283 processor backed by 512MB of flash storage [40].

The Zero provides a lot of room for flexibility with 40 available GPIO pins. The combination of the Zero's computing power and general IO makes it ideal for small spaces that need a lot of power [41].

Summary

The Zero could be useful for programming devices in the system; however, while the Zero does offer a smaller physical footprint and more GPIO pins than the Arduino Uno, it does require more power to maintain operation. This extra power consumption does come with more performance which may be useful but likely unnecessary for the smart learning system.

B-7 Raspberry Pi 1 Model A+

Description

The Raspberry Pi 1 Model A+ is the original Raspberry Pi with some performance improvements. This device is a computing board that provides desktop equivalent computing power in only a few square inches of space. The Pi 1 is the first candidate being considered for the role of the smart home learning hub.

Technical Overview

The Pi 1 is the base Raspberry Pi that is powered by a 32-bit 700MHz Arm processor and 256MB of DDR2 RAM. The Pi 1 comes with a standard HDMI output for visual output. The board also comes with a standard Ethernet port, and a single USB port for serial communication. The Pi 1 does also offer 40 GPIO pins for more embedded purposes [42].

Summary

The Raspberry Pi 1 is a good candidate for the central hub as it uses low power and provides adequate computing performance.

B-8 Raspberry Pi 2 Model B

Description

The Raspberry Pi 2 Model B is the second generation of Raspberry Pi designs. The Pi 2 uses the same design as the Pi 1 with all the same features and more performance.

Technical Overview

The Pi 2 is very similar to the Pi 1 but provides a slight faster 32-bit 900 MHz Arm processor. The Pi 2 has significantly more RAM than the Pi 1 with 1GB of DDR2 [43]. The Pi 2 comes with a standard HDMI video output for monitoring it from a screen. It also is equipped with an Ethernet port and 4 USB ports. The Pi 2 also provides the same 40 GPIO pin configuration as the Pi 1 [44].

Summary

The Pi 2 outperforms the Pi 1 in all areas and is likely a better candidate for the smart hub. It uses the same amount of power but provides far more computing performance.

B-9 Raspberry Pi 3 Model B

Description

The Raspberry Pi 3 Model B is the most advanced Raspberry Pi available. The Pi 3 is a computer board that uses a very similar design to the other Pi Models. The Pi 3 offers more computing performance than all of its predecessors and is a very suitable candidate for the smart hub.

Technical Overview

The Raspberry Pi 3 offers massive embedded performance with a 1.2 GHz 64-bit Quad-core Arm processor [45]. Similar to the Pi 2, the Pi 3 also comes with 1GB of RAM. As with all other Pi models, the Pi 3 provides a 40 GPIO pin configuration for external devices. [46]

The Pi 3 also comes with a number of standard options that set it apart from all other Pi models. It comes with integrated WiFi, Ethernet and Bluetooth communication interfaces [47]. It has 4 standard USB ports and an HDMI output for visual feedback [48].

Summary

The Raspberry Pi 3 is the most advanced Raspberry Pi board available. Its extra computing power would be a good asset for heavy computation making this an ideal candidate for the central smart hub. The Pi 3 comes with many standard features including WiFi and Bluetooth communication which will make external interfacing simple with minimal investment.

B-10 Comparison of Raspberry Pi

Operation Criteria

Criteria	Raspberry Pi Zero	Raspberry Pi 1	Raspberry Pi 2	Raspberry Pi 3
Operating System	Raspbian	Raspbian	Raspbian	Raspbian
Processor Size	32-bit	32-bit	32-bit	64-bit
Processor Family	Broadcom	Arm	Arm	Arm
Operating Voltage	5 V	5 V	5 V	5 V
Input Voltage	3.3 V	3.3 V	3.3 V	3.3 V
Clock Speed	1 GHz	700 MHz	900 MHz	1.2 GHz
Digital Pins	40	40	40	40
Pulse with Modulation Pins	N/A	N/A	N/A	N/A
Analog Input Pins	N/A	N/A	N/A	N/A
DC Current per Pin	50 mA	50 mA	50 mA	50 mA
Flash Memory	512 MB	256 MB	1 GB	1 GB
System Size	1.6 GB	1.6 GB	1.6 GB	1.6 GB

Features

Feature	Raspberry Pi Zero	Raspberry Pi 1	Raspberry Pi 2	Raspberry Pi 3
USB	USB-Micro	1	4	4
HDMI	Mini-HDMI	Yes	Yes	Yes
Bluetooth	No	No	No	Yes
WiFI	No	No	No	Yes
Audio	No	3.5 mm Jack	3.5 mm Jack	3.5 mm Jack
Ethernet	No	Yes	Yes	Yes
Camera Interface	No	Yes	Yes	Yes
Display Interface	No	Yes	Yes	Yes
Micro SD	Yes	Yes	Yes	Yes

Physical Characteristics

Dimension	Raspberry Pi Zero	Raspberry Pi 1	Raspberry Pi 2	Raspberry Pi 3
Length	65 mm	85 mm	85 mm	85 mm
Width	30 mm	56 mm	56 mm	56 mm

B-11 BeagleBone

Description

The BeagleBone is another computing board that is on the edge of embedded computing. The BeagleBone is a small, feature rich, Linux system for concurrent, real-time embedded programming. This board could be used for the learning hub or even a small device that requires more computation power than a microcontroller can offer.

Technical Overview

The standard BeagleBone is very comparable to the Raspberry Pi 2 in features and performance. It is driven by a 32-bit 700MHz Arm processor with 256MB of flash memory. This computing board offers 2 USB ports and a standard Ethernet port for external communication [49].

The BeagleBone stands out from the Raspberry Pi with its general pin configurations, as it offers 60 GPIO pins [50]. This makes the BeagleBone more suitable for embedded computing than the Raspberry Pi models. The BeagleBone is a Linux compatible board that comes with the Angstrom distribution [51].

Summary

The BeagleBone has a number of useful features for embedded computing but does not provide as much computation performance as the Raspberry Pi 3. The BeagleBone may be suitable for other devices in the system that require more computational power than what a traditional microcontroller can provide.

B-12 BeagleBone Black

Description

The BeagleBone Black is a more advanced version of the standard BeagleBone. It provides more features and is a higher performance system than the standard board. The BeagleBone Black is most comparable to the Raspberry Pi 3 board in terms of performance and features. The Black model still offers all of the major BeagleBone embedded features making it a good candidate for both the learning hub and other computing devices

Technical Overview

The BeagleBone Black is driven by a 32-bit 1GHz Arm processor and is backed by 512MB of RAM. It also comes with more standard communication options than is predecessor including WiFi and Bluetooth. The Black model comes standard with 2 USB connections for serial communication and an HDMI interface for video feedback [52].

The Black Model uses the same pin configuration as the basic BeagleBone with a few extra specialty pins. It provides 68 GPIO pins with an additional UART connection [53].

Summary

The BeagleBone Black is does have many powerful features but cannot complete with the Raspberry Pi 3 for the learning hub roll. The Black model may be useful for embedded devices that require additional computation but since it requires more power than the standard BeagleBone with minimal gain it may not be feasible.

B-13 BeagleBone Green

Description

The BeagleBone Green is the best embedded option of the BeagleBone family. It is a computing board that runs a full operating system but has the most available circuit controlling pins. It is an ideal candidate for both the learning hub and smart devices.

Technical Overview

The BeagleBone Green model is powered by the same 32-bit 1GHz processor as the Black model. It also has the same 512MB of RAM available. Almost all of the core features of the Green model are the same as the Black model. The Green model stands out with its 4 additional pulse with modulation pins and 3 additional analog pins. In addition to a UART connection, the Green model provides an I2C connection [54].

Summary

The Green model shares many of the same features as the Black and is therefore not an adequate alternative for the Raspberry Pi 3 and the learning hub. However, the Green does require less energy than the Black model and is possibly a better candidate for embedded devices.

B-14 Comparison of BeagleBone

Operation Criteria

Criteria	BeagleBone	BeagleBone Black	BeagleBone Green
Operating System	Angstrom	Debian	Debian
Processor Size	32-bit	32-bit	32-bit
Processor Family	ARM	ARM	ARM
Operating Voltage	5 V	5 V	5 V
Input Voltage	1.8 V	1.8 V	1.8 V
Clock Speed	700 MHz	1 GHz	1 GHz
Digital Pins	60	68	65
Pulse with Modulation Pins	4	4	8
Analog Input Pins	4	4	7
Flash Memory	256 MB	512 MB	512 MB
System Size	1.8 GB	2.2 GB	2.2 GB

Features

Feature	BeagleBone	BeagleBone Black	BeagleBone Green
USB	2	2	2
HDMI	No	Yes	No
Bluetooth	No	Yes	Yes
WiFI	No	Yes	Yes
Ethernet	Yes	Yes	Yes
UART	No	Yes	Yes
I2C	No	No	Yes
Micro SD	Yes	Yes	Yes

Physical Characteristics

Dimension	BeagleBone	BeagleBone Black	BeagleBone Green
Length	86 mm	86 mm	86 mm
Width	54 mm	54 mm	54 mm

B-15 Summary of Devices

Embedded Devices

There will be a number of applications for embedded devices in the smart home system. There may be various computing devices that are suitable for different tasks depending on the requirements. However, for the general needs of the embedded computing in this system, the Arduino Uno will likely be the most suitable candidate.

The Uno provides a rapid prototyping environment with support for a wide array of custom devices. It has sufficient computing power with minimal power consumption for the requirements of the general devices that have been identified in the system scenarios.

As an added benefit, the Arduino community offers many high quality tutorials, examples and documentation of system usage. The extensive support offered by the community is a major advantage over the other systems and will be a major asset for developing on this device.

Learning Hub

The learning hub will require a significant amount of computational performance to make decisions about home environment. For this component, full computer boards were considered as they provide more performance than the available microcontrollers. After a close examination of a number of computing devices it was determined that the Raspberry Pi 3 Model B is the most suitable option for this roll in the system.

Since this device is heavily reliant on performance, the decision was reduced to 2 candidates, the Raspberry Pi 3 and the BeagleBone Black. They both offer considerable performance but the Pi 3 offers more processing cores with a higher clock speed, larger registers and more RAM. This makes the decision simple, the Pi 3 is the better candidate.

C Communication Protocols

Background

There are many different ways to connect devices across a network. A familiar example is WiFi, but other protocols are available that specialize in different aspects of wireless communication. The goal of this section is to investigate the strengths and weaknesses of different communication protocols, and evaluate them against each other.

Selection Criteria

The criteria used for evaluating communication protocols are based on the non-functional requirements identified for the system. The key requirements pertaining to communication protocols are ease of integration with devices, battery life of devices, range, interoperability, data transfer rate, number of concurrent connections. Another important aspect of a communication protocol that will be evaluated is what frequency it operates on.

C-1 ZigBee

Description

The goal of the ZigBee protocol is to provide a means to transfer small amounts of data over a limited distance. Relative to WiFi, the decreased in data transmission rates and range of ZigBee allows compatible devices to consume less power allowing for increased battery life and energy efficiency. ZigBee shields are available for many popular embedded communication devices, allowing them to communicate with ZigBee compatible smart devices. Many companies offer smart devices which are compatible with the ZigBee protocol, such as lights and light switches.

Technical Overview

ZigBee uses a mesh networking topology, as opposed to the star topology used by WiFi. A mesh topology means that every node in the network is connected. Each node transmits its own information, as well as assisting in relaying information received from other nodes. Having every node connected allows for data to be transmitted between nodes simultaneously, and increases network stability on not relying on one central node. This increase in stability comes at the cost of having potentially many redundant connections in the network. ZigBee devices use the mesh topology to send messages using message routing. This means that if the endpoint device is out of range of the initial device. Intermediate devices will relay the message through the mesh until it reaches its endpoint.

ZigBee operates within three possible frequency bands: 868-870 MHz, 902-928 MHz, and 2.4-2.4835 GHz. The lowest band only has one available channel, the middle band has ten available channels, and the highest band has 16 available channels. The respective data transfer rates are 20 kbps, 40 kbps, and 250 kbps. It should be noted that devices on different frequency bands cannot communicate with each other, and generally only devices using the 2.4 GHz range are commercially available.

The low power consumption of ZigBee devices when compared to WiFi leads to better energy efficiency and a longer batter life.

A device can be added to a ZigBee network in approximately thirty milliseconds. Theoretically, up to 256 devices can be connected to one network. However, in practise, the system performance tends to degrade at around thirty devices.

One of the major drawbacks of ZigBee is that for it to be effective, it must operate in the 2.4 GHz frequency band. This would not be an issue, except for the fact that this is the same frequency band as older versions of WiFi. This can cause interference between the two networks, resulting in packet loss for both networks. The lost packets have to be retransmitted until they are received by the intended endpoint, causing lag in both networks. ZigBee packets suffer more from this interference in practice, with the level of interference rising as the number of nodes and the amount of traffic rises. Fortunately, newer WiFi networks operate on the 5 GHz channel which would eliminate this interference.

A second potential drawback of ZigBee is the historical lack of official standards for application-level protocols for ZigBee devices. While ZigBee devices all communicate using the same physical layer protocol, each device may use a different high-level protocol for device control. The result of this lack of standardization is that different companies have their own protocols for ZigBee device communication, so devices from different companies cannot be assumed to be compatible. The limited interoperability has been somewhat fixed with the introduction of the ZigBee Alliance, but could present some legacy issues.

C-2 Z-Wave

Description

Z-Wave is a very similar option to ZigBee, except it uses a proprietary radio design. This slightly limits the number of devices available for it, as chips are mostly produced by Sigma Designs. The advantage to this is that because the chips are made largely by one manufacturer, there is a high level of interoperability. In addition, the Z-Wave protocol includes high-level commands for controlling certain classes of devices, which further increases interoperability of Z-Wave systems.

Z-Wave goes beyond a network layer protocol and also includes a application-level set of commands for controlling Z-Wave compatible devices. Devices that implement the Z-Wave protocol are standardized and can be controlled using the Z-Wave protocol messages.

Technical Overview

One area where Z-Wave differs from ZigBee is the frequency range of operation. It operates at 908.42 MHz instead of at 2.4 GHz, which avoids the issue of conflicting with WiFi signals. In terms of device limits, it is very similar, being able to handle between 30 and 40 devices before issues start to occur. Z-Wave is similar to ZigBee in terms of device range and power consumption.

C-3 Insteon

Description

Insteon is substantially different from the two above protocols.

Technical Overview

Insteon uses a similar mesh topology to the above protocols, but it is not limited to radio frequencies. It utilizes a dual-mesh system to increase overall stability. The dual-mesh system is a combination of radio frequencies at 915 MHz (in the US), and powerline layer operating at 131.65 KHz.Powerline communication is a technology that uses a home's electrical wiring to transfer data.

When the radio frequencies encounter interference, the powerline layer makes sure the message gets broadcasted to the appropriate destination. Insteon also uses a different message delivery system compared to ZigBee and Z-Wave. Instead of sending a message from one device and routing it through other devices, it takes advantage of simulcasting. This is the process of having multiple devices broadcasting the same message, so the intended recipient gets the message faster and more reliably. This method is not feasible for high data rates, but Insteon shares it's low data rates with ZigBee and Z-Wave. Simulcasting is also a result of the fact that an Insteon network does not have a master/slave relationship. Every node has the ability to send and receive messages without having a controller. This makes it possible to have any number of devices in a network without being restricted by a maximum number of connections to a controlling device.

One thing Insteon is lacking compared to ZigBee and Z-Wave is third party support for their devices. They manufacture almost all of their own devices, which leads to a limited amount of choice in terms of different types of devices designed for the same task.

A unique advantage of Insteon is its ability to interface with devices following the X10 protocol. The X10 protocol was one of the original protocols designed to work using only power lines. It is outdated now in terms of being a reasonable choice for a new system, as it was designed over 30 years ago. This means that there is no wireless communication involved, which is essential for a modern day smart home communication protocol. That said, there are many legacy automation systems in place which still use X10 devices. If this were the case, then Insteon would be an ideal choice for a communication protocol.

C-4 WiFi

Description

WiFi is by far the most common communication protocol used in a home on a daily basis. Nearly every other communication protocol is compatible with WiFi devices, and they tend to be easy to install. There are several pros and cons associated with using WiFi for home automation, which are outlined below.

Technical Overview

WiFi operates using a star network topology. Every node is connected to a central server node. All communications go from the source node, through the central server node, and arrive at the destination node. This means that if the central server goes down, no messages can be passed, leading to potential stability issues. Because messages cannot be routed through intermediate nodes, the communication range must generally be much larger than in a mesh network. WiFi also boasts substantially higher data transmission rates than the communication protocols discussed previously. A WiFi (802.11b) connection can transfer data at a rate of 54 Mbps at a range of 100 meters. As stated earlier, WiFi typically operates at a frequency of 2.4 GHz.

The biggest advantage of using WiFi for home automation is the simplicity. There is virtually no limit to the number of devices that are WiFi compatible, as devices using nearly every other big communication protocol for home automation are also compatible with WiFi. It also requires no extra hardware to communicate with WiFi, which reduces the complexity of setting up hardware. Non-technical users are still generally familiar with WiFi, making the setup process easier.

An issue with using WiFi for home automation is that in terms of required range and data transfer specifications, it is overkill. WiFi is designed with high intensive data transfer in mind, which home automation systems do not take advantage of. A WiFi device needs to provide enough power to operates with these specifications, leading to a large increase in power consumption. If the device is battery powered, it will drain much more quickly than if it were using one of the above protocols.

Another issue stems directly from the prevalence of WiFi. Having a smart home network sharing a WiFi network with regular household uses can cause the network as a whole to slow, due to the bandwidth being shared across so many devices.

C-5 Bluetooth

Description

Bluetooth is the most similar to WiFi of the alternative options. It is fairly common in households, and non-technical users are more likely to be familiar with it than other protocols. There are two main classifications of Bluetooth when it comes to home automation, both of which will be discussed below.

Technical Overview

Bluetooth operates in the 2.4 GHz frequency band, alongside WiFi and ZigBee. Bluetooth also shares the star network topology with WiFi, and is ideal for master and slave devices. This can lead to the same interference problems as discussed in the WiFi and ZigBee sections. As the number of devices on the same radio frequency increases, the competition for bandwidth also increases, causing potential latency and interference. The range and data transfer rate for Bluetooth ranges from 1 Mbps and 10 meters to 24 Mbps and 100 meters. All of these data transfer rates are acceptable for a smart home system. The range on the earlier versions of Bluetooth is potentially very restricting. Bluetooth is somewhere between WiFi devices and ZigBee/Z-Wave devices in terms of power consumption.

There is another choice for Bluetooth that addresses some of the issues above. Bluetooth version 4.0, also branded as Bluetooth Low Energy (BLE). This is a direct competitor with ZigBee and Z-Wave. The range for a BLE device is 50 meters and BLE is able to take advantage of a mesh network topology. The maximum data transfer rate 1 Mbps in theory, but it is generally much lower than that in practise. BLE splits the 2.4 GHz channel into smaller sub-channels to help avoid interference with WiFi channels.

One of the goals of BLE is to make devices that do not require constant data transmission more efficient. It accomplishes this by closing inactive connections while no data is being transferred. Once data needs to be transferred, it reestablishes the necessary connection, completes the transfer, and closes the connection again.

C-6 Summary

Evaluation Criteria

Protocol	Transfer Rate	Battery Life	Interoperability	## of Connections	Frequency	Range	Topology
ZigBee	250 kbps	Good	Good	256	2.4 GHz	35 Ft.	Mesh
Z-Wave	40 kbps	Great	Great	232	915 MHz	100 Ft.	Mesh
Insteon	13 kbps	Good	Bad	N/A	915 MHz	150 Ft.	Mesh
WiFi	54 Mbps	Bad	Great	256	2.4 GHz	105 Ft.	Star
BLE	10 kbps	Good	Great	9	2.4 GHz	200 Ft.	Star

As stated above, the goal of this research was to pick an appropriate protocol for our system. One of the differing attributes between the protocols is the data transfer rates. The required data rate for most smart home devices is minimal, and a higher data rate demands more power. The data rate provided by WiFi is more than required for our project so it will not be the primary communication protocol used. That being said, WiFi compatibility is important to this project because of its prevalence in homes, and because of the enormous amount of devices that communicate over WiFi.

Having as inclusive device support as possible is a key aspect to our project, as it allows a user to have whatever functionality they desire. This heavily influenced us in deciding not to use Insteon as our primary protocol, as the backwards compatibility with X10 power line communication protocol is not something we require.

ZigBee and Z-Wave are very similar, with the key difference in our eyes being the consistent interface that Z-Wave devices provide for controlling them. In additionally, avoiding interference conflicts between the prevalent communication protocol, WiFi, is a benefit. Z-Wave operates in the less used 900 MHz frequency band, avoiding any potential conflicts.

A mesh network topology has many benefits over a star topology for message routing and communication in home automation systems. The stability offered by a mesh topology is aids in reliability, and the amount of data transmission is low enough that the redundant connections are not costly. This makes Z-Wave a more attractive alternative than Bluetooth Low Energy.

Overall, using Z-Wave as our primary communication protocol appears to be the best choice for this project. For industry compliance, WiFi will be required as well.

Z-Wave Implementation Specifics

Any processing unit with USB support can be easily turned into a Z-Wave master, using a Z-Wave USB stick. Converting an Arduino into a Z-Wave slave is not simple. There are specialized Arduino boards that have the Z-Wave protocol built in. Another option is to attach a radio frequency device to an Arduino, and to implement the Z-Wave stack protocol manually. The easiest way to have Z-Wave slave devices is simply not to make them, but to buy them.

D Custom-Built Devices

Background

The purpose of this section is to present our investigation into the process of building various types of smart devices which might be of use for home automation. The purpose of this investigation is to provide information that the team can use to decide which devices will be included in the system to showcase its machine learning capabilities. In particular, this section will examine the feasibility of assembling such devices from basic electronic components, rather than purchasing a commercial device.

In this section, a **smart device** refers to a sensor or actuator which can be controlled over a wireless network. The process of building a smart device is expected to consist of interfacing a hardware module with a microcontroller which is capable of controlling the device and communicating over a network.

Evaluation of custom devices will focus on the following characteristics:

Expertise required

How difficult is it for a team of software engineering students to build the device? Some complex devices will be impractical for the team to build due to the lack of hardware knowledge. Factors which will be considered in the evaluation of the devices include the availability of tutorials and development kits. Safety will also be a critical factor (Does the team have the facilities and expertise necessary to build the device safely).

Level of Effort

In order to allow the team to focus on the machine learning aspects of the system, devices which can be used to demonstrate the system must be available quickly. In order to measure this criteria, the amount of effort required to build an LED controlled through a voltage relay will be used as a baseline for estimation. The following scale will be used:

Low

- Comparable to baseline (less than a day of work)

High

- High effort (A couple days)

Extreme

- Extreme effort (many days or weeks)

Quality Comparisons

Commercially available devices may offer features which could be useful to demonstrate the system, but are difficult to include in a custom-built device. It will be useful to determine whether or not the devices described by most custom-build tutorials are fully featured. In some cases, the features of a typical custom-built device will be compared against several commercial options.

Devices of Interest

This section contains a list of device types which were investigated, and a short explanation of why the devices could showcase the machine learning capabilities of the system. In general, we believe that in order to showcase the machine learning capabilities of the system, it will be important to include a wide range of sensors, as well as actuators whose desired states may be influenced by several unrelated sensors. By providing such a range of devices, we hope to demonstrate that the machine learning algorithm is capable of identifying more complex interactions between devices than a simple IFTTT relationship.

In addition to providing a varied range of sensors, we would like to provide several sensors and actuators with non-binary inputs and outputs. The interactions between such devices may be more complex than for binary devices due to the increased number of possible states.

This section investigates some smart devices which are potentially within the team's ability to build.

Motion Sensors

Many actions could be triggered by a person entering or leaving a house or a room. A motion sensor could potentially interact with every other device on this list. Due to the wide range of potential interactions, we expect that a motion sensor will be a good test of the machine learning component's ability to determine the relationships between inputs and outputs. The motion sensor could also reveal flaws in the machine learning algorithm. For example, the machine learning algorithm may determine that when a motion sensor is triggered outside a homeowner's bedroom, the coffee machine should be turned on. However, this behaviour would only be desired during certain hours of the day (the coffee machine should not turn on when the homeowner goes to bed, for example). It is likely that there are many such scenarios that we haven't thought of which would present challenging cases for the machine learning component to handle.

Thermostat

The setting of a thermostat may be affected by several different factors, such as temperature, time of day, and light levels. Since thermostats are also a non-binary output device, their behaviour when controlled by the machine learning algorithms may be more varied than other devices.

Light Sensor

A light sensor provides a non-binary input to the machine learning component. Similar to a motion sensor, the level of light in a room may be related to the behaviour of many other devices.

Dimmer Switch

A variable-voltage switch is an example of a non-binary output device. A variable-voltage switch could be used to control the brightness of lights, or the speed of a fan.

Coffee Makers

The coffee maker could be turned on in response to several different input devices (light sensor, motion sensor)

D-1 Motion Sensor

Technical Overview

This section considers devices that can be used to alert the system when a person enters or leaves an area of the home.

Sensor Technologies

This section provides a comparison of several different sensor technologies that are commonly used in motion detection.

Passive Infrared (PIR)

A passive infrared sensor detects motion using the infrared radiation (heat) from a warm body. A sensor contains two slots, each of which contains a material which is sensitive to infrared radiation. As a warm body passes in front of the sensor, a different amount of radiation is received at each slot. This difference in radiation causes a signal in the output of the device.

Use of the PIR modules encountered during this research does not require expert knowledge of circuit design; the complexity of the circuits involved in connecting the modules to a microcontroller is comparable to that of a simple LED circuit. A power source and a resistor is sufficient to get the module up and running.

Most PIR modules have 3 pins to connect to a circuit. One pin connects to ground, another to power, andthe third pin is used for output. The output consists of a single digital signal indicating when motion is detected.

Tutorials describing how to connect PIR modules to common microcontrollers, such as a Raspberry Pi or Arduino are widely available online.

The complexity of the circuit required to interface with a PIR module is comparable to the light switch circuit, consisting of a single push button used to toggle an LED. Therefore, development time required for the complete device is expected to be under 10 hours.

PIR sensor modules are reliable because they do not typically contain any moving components, other than potentiometer used for sensitivity adjustment.

PIR sensors are easy to use, the main difficulties that a user may encounter in setting up a PIR sensor are ensuring that the detectorès field of view covers the correct area, as well as potentially adjusting the sensitivity of the detector.

PIR sensors are available with a range which is suitable for a typical room (approximately 7m range).

The following tutorials explain how to interface with PIR modules:

https://cdn-learn.adafruit.com/downloads/pdf/pir-passive-infrared-proximity-motion-sensor.pdf https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor/

Doppler-Effect based sensors

Several types of motion detectors use the Doppler effect to detect motion. The detector transmits a signal of a known frequency. This signal is reflected by any objects in its path and detected when it returns to the sensor. The sensor tracks the frequency of the reflected wave; when a wave is reflected off of a moving object the frequency of the reflected wave differs from that of the incident wave due to the Doppler effect. This change in frequency is detected by the sensor and registered as motion.

Infrared Break-Beam

An infrared break-beam sensor consists of two physical modules separated by some distance. One side of the detector transmits a beam of infrared light which is detected by the other side.

Expertise Required

- Little expertise required, circuits are simple
- Tutorials widely available online

Component Availability

- Components widely available

Reliability

- Sensors generally have a lower range than PIR sensors, availability of detectors with a longer range may be lower
- Allows for finer control over the area of detection than a PIR sensor

Usability

- More difficult setup than a PIR sensor, requires the user to precisely aim the transmitted beam.

Feature Comparison

Feature	Custom	D-Link Wifi Motion Sensor	Samsung SmarttThings Motion Sensor	Belkin
Range	7m	8m	15m - 40m	3m
Interfaces	_	WiFi	ZigBee	WiFi
Type	PIR	PIR	PIR	PIR

Note: The custom device can potentially support multiple different communication interfaces

Evaluation

Criterion	Score
Expertise Required	Low
Level of Effort	Low
Quality Comparison	Equal

D-2 Variable-Voltage Switch

The characteristics of variable-voltage switches vary significantly depending on the type of device that the switch is expected to control (DC vs AC).

Controlling DC power

If the type of device being controlled requires DC power input, the amount of power supplied to the device can be controlled using pulse-width modulation (PWM). A digital output is used to create a square wave. By varying the frequency of the square wave, it is possible to simulate the application of a steady voltage between the pin's high and low voltages.

Limitations to simple PWM

- Power available to the device is limited by the power supplied by the controlling device - Limited to DC devices

Expertise Required

Minimal expertise is required to use PWM to control a device. The circuits required are as simple as the circuits required to power a simple LED.

Reliability

Simple PWM circuits are limited to controlling devices with low power requirements. While the technique is reliable for devices which meet the power requirements, such as LED lamps, the power requirements of most home devices means that the applicability of this technique will be severely limited. Most devices are intended to be connected to a wall socket, meaning that they are expecting AC current, which a significantly higher amount of power than is available from a microcontroller such as an Arduino.

Controlling AC power

Expertise Required

- Safety concerns: Controlling devices which expect to receive the voltages available from a wall socket means dealing with high voltages. Without any team members trained in using high voltages, building a device to control high voltages is a very serious safety risk.
- Lack of trusted tutorials: For most of the circuits considered during this research, it has been possible to find tutorials online from trusted sources, such as the official Arduino website.

Level of Effort

- When it was possible to find a tutorial online for building a circuit to control AC voltages, the circuit was found to be much more complex than the simple light switch circuit. Due to the amount of time required to fully understand these circuits, it is estimated that effort required is high.

Feature	Custom	GE Z-Wave Plugin-in Smart Dimmer	Philips Hue Dimmer Switch	Lutron Caseta Wireless Plugin-In Lamp Dimmer
Compatible Devices	AC devices	AC devices	Philips Hue Products	AC Devices
Interfaces	_	Z-Wave	ZigBee	Lutron Integration Protocol

Evaluation

Criterion	Score
Expertise Required	High
Level of Effort	High
Quality Comparison	Equal

Note: There are safety concerns for custom-building this device

D-3 Light Sensor

Description

This section investigates devices which can detect the amount of ambient light in an area of the house.

Photoresistors

Light sensors generally make use of a component called a photoresistor. The resistance of a photoresistor differs depending on the amount of light incident to the component. Phtoresistors are also called photocells.

Reliability Characteristics

- Photoresistors are unsuitable for precise lighting measurements because its resistance may vary due to temperature as well as light
- Photoresistors may exhibit latency (delay between a change in light and a change in resistance)

Photodiode

Photodiodes are components which can be used similarly to photoresistors to detect light. A photodiode allows electrons to pass when there is light shining on it. The current allowed to pass is proportional to the amount of light incident on the device.

Unlike photoresistors, a photodiode is not sensitive to temperature changes and may allow for more accurate light detection. The circuit required to interface with a photodiode is of similar complexity to the circuit required to interface with a photoresistor.

Expertise Required

Minimal expertise is required to build the devices described by most guides and tutorials. The circuit is not significantly more complex than the light switch.

Level of Effort

Due to the simplicity of the photoresistor circuit and the wide availability of the required components and tutorials, a custom-built light sensor is required requires a low amount of effort.

Feature Comparison

Many commercially available light sensors are included in combination devices which include several different types of sensors. Commercial solutions provide very little data about the range of light that their products can detect. For a custom built solution, the range of detectable light depends on combination of the particular photocell chosen. Assuming that both a custom device and most commercial devices are able to differentiate between daytime and evening levels of light, additional features available from commercial devices are generally related to other types of sensors

Evaluation

Criterion	Score
Expertise Required	Low
Level of Effort	Low
Quality Comparison	Equal

D-4 Thermostat

This section considers controlling a typical wall-mounted thermostat using a microcontroller such as an Arduino.

Expertise Required

Safety: Controlling a wall-mounted thermostat means working with mains voltage levels. Without knowledge in using high voltages, building this device is a significant safety risk

Effort Level

Due to the lack of trusted tutorials online, as well as the wealth of features that users are used to having in a thermostat, the time investment required to build a fully-featured thermostat is extreme.

Evaluation

Due to the extreme level of effort required and the safety concerns with building a thermostat, a custom-built thermostat is not feasible for this project.

D-5 Alarm Clock

There are a large variety of features available for alarm clocks, this section will focus on a simple alarm clock that beeps at a programmed time of day. The clock should have manual (button) input in order to allow the user to set alarm times so that the machine learning component can be trained. More sophisticated clocks are expected to be much too complex for the team to build.

After review of many online tutorials for building alarm clocks, a common structure for a clock circuit has been identified.

The basic alarm clock circuit generally requires the following components:

- LCD display
- Microcontroller
- Piezo Buzzer or Speaker
- Varying numbers of push buttons for manual configuration

Level of Effort

Due to the need to interface with an LCD display, and because of the large number of components, a custom build of an alarm clock is expected to be more complex than the light switch. The level of effort is estimated to be high because of this.

Feature Comparison

Homeowners may be used to alarm clocks with nice interfaces, often in the form of an app on their phone. The custom alarm clocks investigated have significantly inferior interfaces, making use of a 4-button interface to set the time. The clocks considered here also lack features such as adjustable alarm sounds, configuring multiple alarms, and setting weekly alarm schedules. Making a custom built alarm clock as usable as commercially available clocks would require significant time investment.

D-6 Coffee Makers

Using a Relay Switch

A simple way to connect a coffee machine to a microcontroller is by using a voltage relay. A voltage relay is an electrically controlled switch for high power devices. A simple coffee machine that only needs to be plugged in in order to start brewing could be controlled using a simple voltage relay circuit.

Level of Effort

The same circuit which is used in the LED switch could be used to control the coffee machine, since both the coffee machine and LED can be connected to the same voltage relay, so the level of effort required is low.

Required Expertise

- Safety risks: Coffee machine are normally connected directly into a wall socket. Controlling the coffee machine using a voltage relay would therefore involve high voltage levels which are unsafe to work with without proper training

In order to interface a microcontroller with more sophisticated coffee machines, some level of reverse engineering of the coffee machine's control circuits would be necessaary. This would greatly increase the amount of time necessary to create the device, due to the lack of electronics knowledge on the team.

Evaluation

Custom builds of both simple and fully-featured coffee makers may be infeasible due safety concerns when controlling high voltage devices.

D-7 Summary

Device	Level of Effort	Safety Concerns	Fully-Featured
Motion Sensor	Low	None	Yes
Light Sensor	Low	None	Yes
Dimmer Switch	Low	High Voltage	Yes
Coffee Maker	High	High Voltage	No
Thermostat	High	High Voltage	No
Alarm Clock	Low	None	No

Conclusions

It is feasible for the team to build motion sensor, light sensors, and an alarm clock if no acceptable commercial solution is available. We should avoid building dimmer switches, coffee makers, and thermostats due to the potential high-voltage work involved and the lack of electronics knowledge available.

E Smart Devices

Background

In the context of this project, a smart device is a device that is capable of communicating over a wireless computer network, such as WiFi or Z-Wave. Each device provides a set of inputs and outputs which can be controlled and examined using some protocol.

Relation to System

This section examines several commercial smart device products which are available off-the-shelf. The goal of this research is to determine what types of devices are commercially available, and to compare alternative products which have similar functionality. By purchasing off-the-shelf smart devices, the development time for the project can be focused on building the machine learning features, which differentiate our product from many existing automation systems. An awareness of existing smart devices will also allow us to select common technologies to support in the system. Support for commercial smart devices makes our system more appealing to the end user because they are not limited to only using devices we create.

Where possible, research will focus on answering the following questions about a product or line of products:

- What inputs and outputs does the device have? What type of information can the device provide to the machine learning algorithm, and what can be controlled?
- Which communication protocol(s) are supported by the device?

- Are the interfaces used to communicate with the device well-documented? Are there tutorials and SDKs available?
- What restrictions does the device introduce to the system? For example, do users need to create an account with another company in order to use the device with our system?

E-1 WeMo

Description

WeMo is a line of smart devices made by Belkin. WeMo devices can be controlled over a WiFi connection using a smartphone app.

Available Devices

1. Light switch

- On/Off

1. Outlet switch

- On/Off

1. Camera

- On/Off
- Motion detection

1. LED lights

- On/Off
- Dimming

1. Slow Cooker

- On/Off
- Temperature Control

1. Coffeemaker

- On/Off
- Brew status
- Change filter

1. Air Purifier

- Fan speed
- Ionizing On/Off
- Filter life

1. Humidifier

- Humidity level
- Water status
- Filter status

1. Heater

- On/Off
- Temperature
- Power level

Technical Overview

Communication Protocol

WeMo only supports WiFi for communicating with devices. No central hub is required in order to control WeMo devices, each device connects to a WiFi network directly.

WeMo uses Universal Plug And Play (UPnP) protocol for discovery and operating the devices.

Developing with WeMo

- *ouimeaux* is an Open source python API for controlling WeMo devices.
- WeMo devices can be controlled using UPnP. This allows us to implement one protocol and comunicate with all WeMo devices as well as any other smart devices from other vendors that use UPnP.

Research Criteria

All WeMo Devices

Inputs	Outputs	Developer Support	Protocol	API Restrictions
On/Off	On/Off	ouimeaux library	WiFi	Local Only
			UPnP	We need to Write it

E-2 Nest Thermostat

Description

The Nest thermostat is a smart learning thermostat. Over time the thermostat learns the homeowner's routine and adjusts the temperature accordingly. The thermostat also turns off automatically when it detects that nobody is home, and can be remotely controlled using a smartphone app.

Technical Overview

Nest communicates using WiFi and a custom nest protocol called **Nest Weave**. Nest Weave uses WiFi and the Thread protocol.

Nest provides API support through the nest cloud. The API is accessed as a RESTful service or using Firebase.

Evaluation

The nest thermostat already has some machine learning on it. Trying to control it using our algorithm could cause unexpected results. They also require the use of a cloud API to control the thermostat and we are trying to avoid this and communicate locally with our devices. Due to these resons it seems like the nest thermostat is not a great fit for our system and we should not invest time into integrating with it.

E-3 Honeywell VisionPro Thermostat

Description

The Honeywell VisionPro Thermostat is a smart programmable thermostat which can be controlled using the Z-Wave protocol. Unlike the Nest thermostat, the VisionPro does not include learning features.

Technical Overview

Communication

The thermostat communicates can be controlled through the touch screen or through Z-Wave.

API

There is no API provided by Honeywell. The thermostat is controllable using the Z-Wave protocol.

Evaluation

This is a simple thermostat that will be easy to control using Z-Wave and as it doesn't have any learning on it, so there won't be any conflicts with our system.

E-4 Wireless Speakers

Available Devices

Speakers that have the capability to provide actions performed to our communication hub are required to automate music in a home. There does not appear to be any Z-Wave ready speakers available currently, and existing WiFi speakers do not provide an API to interact with them. The result of this is that appropriate speakers for use in our system do not currently exist.

Developing Wireless Speakers

It could be possible to create our own smart speakers for use in the ARIA system, but it would require a large amount of effort. Development would involve using an Arduino or similar microcomputing device to turn a speaker into a device compatible with our system. This implies that the speaker must be able to communicate using our own defined protocol. Implementing this is not a priority for our project, as the value added is limited, but it can be considered in the future if there is necessary need. To include smart speakers in our home automation system we will need to explore developing our own custom devices.

E-5 Smart TV

Available Devices

Similar to the wireless speakers, there are no available TVs that provide the required functionallity to be a useful device in our system. Creating custom TV devices is also non-feasbile,as the level of complexity for interfacing with a TV and the corrosponding cable box exceeds the capibilities of our team members.

E-6 Philips Hue

Available Devices

1. White Bulbs

- On/Off
- Dimming

1. White Ambiance Bulbs

- On/Off
- Dimming
- Supports multiple shades of white

1. Colour Ambiance Bulbs

- On/Off
- Dimming
- Configurable colour

1. Lightstrips

- Adhesive strips of LED lights
- Dimming
- Configurable colours
- Dynamic lighting/colour schemes
- White light not supported

1. Lightstrip Plus

- Adhesive strips of LED lights
- Dimming
- Configurable colours
- Dynamic lighting/colour schemes
- Supports white light
- Brighter than regular Lightstrips (1600 Lumen)
- Fine-grained fading control

1. Dimmer switch

- Battery-powered
- Doesn't require the Hue bridge
- Can't use it with other AC devices, no electrical contact with bulbs

1. Tap Switch

- Powered by touch - no battery or wires
- Hue bridge and app required

1. Hue Motion Sensor

- Detects motion in the vicinity of a PIR sensor
- Includes an integrated daylight sensor

Developing with Hue

- Devices use ZigBee Light Link
- The Hue bridge is a hub device that allows lights to be controlled using WiFi. It bridges the ZigBee protocol used by lights to Wifi used by apps.
- Bridge works with most standard ZigBee lights
- Hue Bridge provides a RESTful API for controlling connected lights. API is only accessible when you're on the same LAN as the bridge.

Research Criteria

Inputs and outputs differ by device type; developer support, communication protocol, and API restrictions are common to all Philips Hue devics.

Hue Lights

Inputs	Outputs	Developer Support	Protocol	API Restrictions
On/Off	On/Off	Tutorials on Hue website	ZigBee	Local Only
Brightness	Brightness	Android, Java, IOS Official SDKs		REST API requires a hub
Hue	Hue	Numerous 3rd party SDKs		
Saturation	Saturation			
Colour Temperature	Colour Temperature			
Dynamic Effect	Dynamic Effect			

Hue Motion Sensor

Inputs	Outputs
sensitivity	presence
	light level

Hue Dimmer Switch, Hue Tap

Inputs	Outputs
	button event

E-7 Osram LIGHTIFY

Description

LIGHTIFY is a line of lighting products which are can be controlled using the LIGHTIFY mobile app. LIGHTIFY provides two separate product lines; LIGHTIFY Pro and LIGHTIFY Home. LIGHTIFY Pro products are designed to be highly scalable for office environments. This research focuses on the LIGHTIFY Home line of products.

The LIGHTIFY system consists of a gateway device which connects to all of the bulbs installed in the home. Using the LIGHTIFY app, a homeowner can control connected lights from a mobile device.

Available Devices

1. Surface Light TW

- Dimmable
- Adjustable colour temperature
- White only

1. Surface Light W

- Dimmable
- White only

1. Flex RGBW

- RGB colour control
- Adjustable colour temperature
- Dimmable

Technical Overview

LIGHTIFY products use the ZigBee protocol for communication between the gateway and lighting products. The gateway connects to a local Wifi network, allowing the LIGHTIFY app to control the system over the Internet.

Due to the use of the open ZigBee protocol, LIGHTIFY products can be controlled using any ZigBee controller

The LIGHTIFY gateway also provides a RESTful API for controlling lights over internet. One notable limitation of the LIGHTIFY API is that it is a cloud-only API. This means that the LIGHTIFY gateway is strongly tied to a homeowner's LIGHTIFY account; Osram does not document any local-only API for controlling devices using a gateway

Research Attributes

Inputs	Outputs	Developer Support	Protocol	API Restrictions
on/off	on/off	REST API description	ZigBee	REST API is Cloud Only
colour	colour	Sample Application		REST API requires LIGHTIFY account
colour temperature	colour temperature	No Official SDKs		REST API requires a hub
brightness	brightness			
saturation	saturation			
transition time (effects)	transition time			

E-8 Aeotec Light Bulbs

Description

Aeotec sells light bulbs which are compatible with the Z-Wave protocol. Aeotec products are compatible with most Z-Wave hubs. Aeotec does not provide a proprietary API for their products.

Available Devices

1. LED Bulb

- Dimmable
- Configurable Colour

1. LED Strip

- Dimmable
- Configurable Colour

Research Attributes

Inputs	Outputs	Developer Support	Protocol	API Restrictions
on/off	on/off	Not Aeotec Specific	Z-Wave	None
colour	colour	Z-Wave Developer		
brightness	brightness			

E-9 Aeon Labs

Description

Aeon Labs is a company that produces a large variety of Z-Wave devices. They also provide the ability to create your own home automation hub.

Technical Overview

Aeon Labs produce a USB dongle that allows you to turn any computing device into a Z-Wave communication hub. This hub allows the user to manually control any Z-Wave device on the Z-Wave network, and provides the functionality to add and remove devices to/from the network. To add a Z-Wave device, start by unplugging the dongle from the hub and pushing the button on it. Then walk to the new device and push the button on the device. The dongle must also be unplugged to remove a device from the network. To put it in removal mode, push and hold the button on the dongle. Then go to each device you wish to remove from the network and push the button. The dongle can be bought for about $60.

They do not provide their own hub, which is ideal for out project. They simply allow the ability to turn a computing device into a custom hub by providing the required RF signal to communicate with Z-Wave devices. To begin development for Z-Wave devices, The list of Z-Wave devices from Aeon Labs alone is fairly extensive. The list include but is not limited to: door sensors, window sensors, lights, energy meters, range extenders.

The specification for interacting with Z-Wave devices is public, and available at http://zwavepublic.com/specifications. There is also an open Z-Wave sdk available, to communicate from the controller to devices.

Available Devices of Interest

1. MultiSensor

- Motion Sensor
- Temperature Sensor
- Light Sensor
- Humidity Sensor
- Vibration Sensor
- UV Sensor

The MultiSensor is a Z-Wave device, meaning that it follows the Z-Wave protocol for commands interacting with each individual sensor. As stated above, the speficication for interacting with Z-Wave devices is available at http://zwavepublic.com/specifications.

Individual Z-Wave sensors are also available, and function using the same specifications as the MultiSensor above. AARTech is one alternative which has a selection of individual sensors if the MultiSensor is unnecessary.

1. Door / Window Sensor

- Detect window/door state The Door/Window sensor is purposed to be able to detect the state of doors or windows. This allows the system to use information about the state of doors and windows to make decisions about what actions to take.

Evaluation

Aeon Labs is an ideal solution for our project. To start, there is an easy way to set up our own hub, instead of being constrained to buying one. This allows us to implement the features we want, and add support for protocols we want without restriction. Having access to an sdk is very important to us as well. It lets us not be responsible for implementing the Z-Wave stack protocol, while also giving us the freedom to use the information received by the devices in a unique way. Specifically, this will provide us with data for the machine learning algorithm.

E-10 Passive Infrared Sensor

Description

A passive infrared sensor (PIR) is a device that can be used to detect if there are people in a room or not. They are available to buy Z-Wave ready, and can be used as a motion sensor in a home environment.

Technical Overview

A PIR creates a binary output based on if it the sensor is being triggered or not. This makes it useful as a motion detector, as well as being able to detect if people remain present in a room. It is also feasible easy to develop our own PIR sensors, which is discussed in the Custom Devices section of the Appendix.

Evaluation

These devices will be useful in our system whenever it is necessary to detect the presence of a person in an area. There is a large selection of PIRs available to buy that are Z-Wave compatible, and it is also an option to construct our own.

E-11 Spruce Irrigation

Description

Spruce irrigation is a home plant watering automation system. The irrigation system can be used outdoors for watering gardens, lawns or any other plant life. The Spruce system is ideal for scheduling or automating plant watering cycles. The Spruce system also offers automation of water based on weather conditions and soil moisture.

Technical Overview

The Spruce irrigation system uses a combination of wireless sensors, smart sprinklers and a central hub for controlling the watering levels of its plants. The irrigation system's central hub is connected to each of the smart sprinklers to control the water flow. Smart sensors are then placed in various locations on the property and communicate wirelessly to the central hub. The sensors are placed in the ground and are battery powered.

The Spruce hub provides manual control of all the sprinklers in the system. The hub also provides a user interface for scheduling sprinkler times and amounts. The Spruce system is also equip with some learning software that uses weather predictions and moisture amount to optimize water use for watering plants.

Evaluation

The Spruce system provides integration to SmartThings but has no other public APIs. This means that it is not accessible other systems for communication. The only option for controlling the Spruce system would be to implement the SmartThings protocol or to attach an adapter to an existing SmartThings Hub. This is likely onside of the scope of this project but could be pursued if there was interest.

E-12 OSO PlantLink

Description

PlantLink is a indoor or outdoor plant monitoring and irrigation system. PlantLink uses a simple monitoring system where sensors are placed it the target plant's soil for monitoring. These sensors then report back to a mobile app for monitoring. PlantLink also offers an automated valve for controlling water flow of a hose.

Technical Overview

PlantLink sensors are low power, light weight units that can transmit data with a range of 100-300 ft. The sensor is water resistant and battery powered with an estimated one year battery life. The sensors are used to check soil moisture of the plants they are monitoring. The sensors then communicate to your mobile device using email, text or push notifications.

PlankLink also offers a smart valve system for controlling water flow of a hose. The valve system is solar powered and can be controlled from the PlankLink mobile app. The valve can also communicate directly to senors to automate the watering of plants. In this mode, if a sensor reports that the soil is dry then the valve will open up and water the plants.

Evaluation

PlantLink is compatible with other platforms but does not offer open source access to its APIs. PlantLink can communicate with SmartThings, Iris, greenIQ and GRO automation systems. To be able to integrate with this system we would need to interface through one of these other systems as an intermediary. The PlantLink system may be outside of the scope of this system unless there we plan on integrating with another smart system

E-13 Summary of Evaluation

Existing devices with an element of machine learning already incorporated in them are not a good fit for our system, as they may cause unexpected results when introduced to our custom machine learning algorithm. Similarly, devices that do not provide an API are also not suitable for use in our system. Without being able to communicate with a device, there is no way for us to control is from our hub or to retrieve data from it for the machine learning algorithm. These aspects rule out devices such as the NEST thermostat and the irrigation systems.

Third party Z-Wave devices are suitable for our system because they provide an easy method of communication in a way that helps enable the machine learning rather than hinder it. Any device that has no special capabilities other than being Z-Wave ready are ideal, because they allow us to use them with no unexpected behaviours. The Honeywell VisionPro Thermostat and devices from Aeon Labs are examples of this.

F Purchases List

After all of the research that we have done in the previous sections we have decided that we will be purchasing the following device to work with our scenarios.

Device	Protocol	Price	Quantity
WeMo Coffe Maker	WiFi	$150	1
Aeotec Z-Wave RGB Light	Z-Wave	$75	2
PIR	Z-Wave	$45	4
Honeywell Thermostat	Z-Wave	$145	1

After the research we have decided to build the following devices:

- Hub
- Light sensor
- Smart speaker system

The required components to build these devices are listed below.

Component	Price	Quantity
Z-Wave usb stick	$65	1
Raspberry pi	$40	2
Arduino uno	$30	1
Arduino wifi shield	$35	1
Photo Resistor	$5	1

G Raw Data

G-1 Z-Wave Device Data

A full representation of the python object for a zwave device from the openzwave library is below.

```
basic:4
product_name:LB60Z-1 Dimmable LED Light Bulb
capabilities:{
   'routing':0,
   'listening':0,
   'beaming':0
}
command_classes:{
   0,
   32,
   133,
   38,
   39,
   134,
   112,
   114,
   115,
   90,
   94
}
command_classes_as_string:{
   'COMMAND_CLASS_VERSION',
   'COMMAND_CLASS_ASSOCIATION',
   'COMMAND_CLASS_CONFIGURATION',
   'COMMAND_CLASS_POWERLEVEL',
   'COMMAND_CLASS_NO_OPERATION',
   'COMMAND_CLASS_SWITCH_MULTILEVEL',
   'COMMAND_CLASS_MANUFACTURER_SPECIFIC',
   'COMMAND_CLASS_BASIC',
   'COMMAND_CLASS_SWITCH_ALL',
   'COMMAND_CLASS_DEVICE_RESET_LOCALLY',
   'COMMAND_CLASS_ZWAVE_PLUS_INFO'
},
device_type:Light Dimmer Switch
generic:17
groups:{
   1:{
      'label':'Lifeline'
   }
}
neighbors:{
   1:0
}
product_type:0x4754,
values:{
   72057594093076481:{
      'label':'Level',
      'data':99,
      'genre':'User',
      'value_id':72057594093076481,
      'units':'',
      'node_id':3
   },
   72057594102726788:{
      'label':'Test Status',
      'data':'Failed',
      'genre':'System',
      'value_id':72057594102726788,
      'units':'',
      'node_id':3
   },
   72057594102382614:{
      'label':'InstallerIcon',
      'data':1536,
      'genre':'System',
      'value_id':72057594102382614,
```

```
      'units':'',
      'node_id':3
   },
   72057594101465153:{
      'label':'Start Level',
      'data':0,
      'genre':'System',
      'value_id':72057594101465153,
      'units':'',
      'node_id':3
   },
   72057594101481476:{
      'label':'Switch All',
      'data':'On and Off Enabled',
      'genre':'System',
      'value_id':72057594101481476,
      'units':'',
      'node_id':3
   },
   72057594102726673:{
      'label':'Timeout',
      'data':0,
      'genre':'System',
      'value_id':72057594102726673,
      'units':'seconds',
      'node_id':3
   },
   72057594098483220:{
      'label':'Dim Level Memory',
      'data':'Full Brightness',
      'genre':'Config',
      'value_id':72057594098483220,
      'units':'',
      'node_id':3
   },
   72057594102726742:{
      'label':'Frame Count',
      'data':0,
      'genre':'System',
      'value_id':72057594102726742,
      'units':'',
      'node_id':3
   },
   72057594103037975:{
      'label':'Protocol Version',
      'data':'3.95',
      'genre':'System',
      'value_id':72057594103037975,
      'units':'',
      'node_id':3
   },
   72057594093076504:{
      'label':'Bright',
      'data':False,
      'genre':'User',
      'value_id':72057594093076504,
      'units':'',
      'node_id':3
   },
   72057594102726724:{
      'label':'Test Powerlevel',
      'data':'Normal',
      'genre':'System',
      'value_id':72057594102726724,
      'units':'dB',
      'node_id':3
   },
   72057594102726806:{
      'label':'Acked Frames',
      'data':0,
      'genre':'System',
      'value_id':72057594102726806,
      'units':'',
      'node_id':3
```

```
    },
    72057594102382593:{
       'label':'ZWave+ Version',
       'data':1,
       'genre':'System',
       'value_id':72057594102382593,
       'units':'',
       'node_id':3
    },
    72057594102382630:{
       'label':'UserIcon',
       'data':1536,
       'genre':'System',
       'value_id':72057594102382630,
       'units':'',
       'node_id':3
    },
    72057594103037991:{
       'label':'Application Version',
       'data':'5.08',
       'genre':'System',
       'value_id':72057594103037991,
       'units':'',
       'node_id':3
    },
    72057594093076520:{
       'label':'Dim',
       'data':False,
       'genre':'User',
       'value_id':72057594093076520,
       'units':'',
       'node_id':3
    },
    72057594103037959:{
       'label':'Library Version',
       'data':'3',
       'genre':'System',
       'value_id':72057594103037959,
       'units':'',
       'node_id':3
    },
    72057594101465136:{
       'label':'Ignore Start Level',
       'data':True,
       'genre':'System',
       'value_id':72057594101465136,
       'units':'',
       'node_id':3
    },
    72057594102726705:{
       'label':'Test Node',
       'data':0,
       'genre':'System',
       'value_id':72057594102726705,
       'units':'',
       'node_id':3
    },
    72057594102726696:{
       'label':'Set Powerlevel',
       'data':False,
       'genre':'System',
       'value_id':72057594102726696,
       ' units':'',
       'node_id':3
    },
    72057594102726776:{
       'label':'Report',
       'data':False,
       'genre':'System',
       'value_id':72057594102726776,
       'units':'',
       'node_id':3
    },
    72057594102726760:{
```

```
      'label':'Test',
      'data':False,
      'genre':'System',
      'value_id':72057594102726760,
      'units':'',
      'node_id':3
   },
   72057594102726660:{
      'label':'Powerlevel',
      'data':'Normal',
      'genre':'System',
      'value_id':72057594102726660,
      'units':'dB',
      'node_id':3
   }
}
node_id:3
location:
name:
num_groups:1
object_id:3
outdated:True
product_id:0x3038
product_name:LB60Z-1 Dimmable LED Light Bulb
product_type:0x4754
query_stage:Complete
role:Always On Slave
security:0
specific:1
type:Light Dimmer Switch
use_cache:True
manufacturer_id:0x014f
manufacturer_name:Linear
max_baud_rate:40000
name:
neighbors:{
   1
}
home_id:4036346586
is_awake:True
is_beaming_device:True
is_failed:False
is_frequent_listening_device:False
is_info_received:True
is_listening_device:True
is_locked:False
is_ready:True
is_routing_device:True
is_security_device:False
is_sleeping:False
is_zwave_plus:True
kvals:{

}
last_update:None
```

6 References

[1] Apple, "Creating homes and adding accessories," 2016. [Online]. Available: https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/HomeKitDeveloperGuide/WritingtotheHomeKitDatabase/WritingtotheHomeKitDatabase.html#//apple_ref/doc/uid/TP40015050-CH4-SW1. Accessed: Oct. 10, 2016.

[2] E. Betters, "Apple HomeKit and home app: What are they and how do they work?," Pocket-lint, 2016. [Online]. Available: http://www.pocket-lint.com/news/129922-apple-homekit-and-home-app-what-are-they-and-how-do-they-work. Accessed: Oct. 10, 2016.

[3] B. International, "WeMo," WeMo, 2014. [Online]. Available: http://www.belkin.com/whatiswemo/. Accessed: Oct. 6, 2016.

[4] "Ouimeaux 0.8: Python package index,". [Online]. Available: https://pypi.python.org/pypi/ouimeaux. Accessed: Oct. 6, 2016.

[5] "Meet the nest learning thermostat," Nest Labs, 2016. [Online]. Available: https://nest.com/ca/thermostat/meet-nest-thermostat/. Accessed: Oct. 24, 2016.

[6] [Online]. Available: https://developers.nest.com/documentation/cloud/data-structure-and-access. Accessed: Oct. 24, 2016.

[7] Honeywell. [Online]. Available: http://library.ademconet.com/MWT/fs2/5800ZBRIDGE/ZWSTAT-Dealer-Data-Sheet.pdf. Accessed: Oct. 26, 2016.

[8] "Philips hue API," 2014. [Online]. Available: http://www.developers.meethue.com/philips-hue-api. Accessed: Oct. 29, 2016.

[9] P. L. B, "Philips hue,". [Online]. Available: http://www2.meethue.com/en-ca/. Accessed: Oct. 29, 2016.

[10] "OSRAM rest API," 2015. [Online]. Available: https://us.lightify-api.org. Accessed: Oct. 29, 2016.

[11] "Z-Stick 2E manual," Aeotec, Aeon Labs, 2012. [Online]. Available: http://aeotec.com/Z-Wave-usb-stick/913-z-stick-manual-instructions.html. Accessed: Oct. 13, 2016.

[12] "Home automation products," in Aeotec, Aeon Labs, 2006. [Online]. Available: http://aeotec.com/homeautomation. Accessed: Oct. 13, 2016.

[13] "OpenZWave library," in OpenZWave. [Online]. Available: http://www.openzwave.com/dev/index.html. Accessed: Oct. 13, 2016.

[14] "Spruce - how it works,". [Online]. Available: http://spruceirrigation.com/How. Accessed: Oct. 13, 2016.

[15] "Spruce irrigation controller & sensor," SmartThings Support. [Online]. Available: https://support.smartthings.com/hc/en-us/articles/208053773-Spruce-Irrigation-Controller-Sensor. Accessed: Oct. 13, 2016.

[16] "PlantLink," PlantLink, 2016. [Online]. Available: https://myplantlink.com/in-action. Accessed: Oct. 13, 2016.

[17] R. Crist, "Best smart home devices of 2016," CNET, 2016. [Online]. Available: https://www.cnet.com/topics/smart-home/best-smart-home-devices/. Accessed: Oct. 6, 2016. 66

[18] "What Technology?," in SMARTHOME® - Home Automation Superstore, 1995. [Online]. Available: http://www.smarthome.com/sc-what-technology. Accessed: Oct. 6, 2016.

[19] L. LABS, "Z-Wave vs. Zigbee," in Wireless Technology, Link Labs, 2015. [Online]. Available: http://www.link-labs.com/z-wave-vs-zigbee/. Accessed: Oct. 6, 2016.

[20] L. LABS, "The ZigBee vs WiFi battle for M2M communication," in IOT Networks, Link Labs, 2015. [Online]. Available: http://www.link-labs.com/zigbee-vs-wifi-802-11ah/. Accessed: Oct. 6, 2016.

[21] P. Sparrow, "Mesh Topology: Advantages and disadvantages," in I Answer 4 U. [Online]. Available: http://www.ianswer4u.com/2011/05/mesh-topology-advantages-and.html#axzz4MJR54MC7. Accessed: Oct. 6, 2016.

[22] J. Gracia Castro and Ó. Pérez Domínguez, "ZigBee: IEEE 802.15.4," in Tampere University of Engineering, 2007. [Online]. Available: https://www.cs.tut.fi/kurssit/TLT-6556/Slides/4-802.15ZigBee.pdf. Accessed: Oct. 6, 2016.

[23] W. Mardini, Y. Khamayseh, R. Jaradatand, and R. Hijjawi, "Interference Problem between ZigBee and WiFi," 2012. [Online]. Available: http://www.ipcsit.com/vol30/024-ICNCS2012-G3061.pdf. Accessed: Oct. 6, 2016.

[24] "What is Z-Wave," in SMARTHOME® - Home Automation Superstore, 1995. [Online]. Available: http://www.smarthome.com/sc-what-is-zwave-home-automation. Accessed: Oct. 6, 2016.

[25] "Z-Wave vs. Zigbee," in Link Labs, Link Labs, 2015. [Online]. Available: http://www.link-labs.com/z-wave-vs-zigbee/. Accessed: Oct. 6, 2016.

[26] L. Frenzel, "What's the difference between ZigBee and Z-Wave?," in Electronic Design, 2012. [Online]. Available: http://electronicdesign.com/communications/what-s-difference-between-zigbee-and-z-wave. Accessed: Oct. 6, 2016.

[27] "WHITEPAPER: Compared," in INSTEON. [Online]. Available: http://cache.insteon.com/pdf/INSTEONCompared.pdf. Accessed: Oct. 6, 2016.

[28] "X10," in Build Your Smarthome, 2014. [Online]. Available: http://buildyoursmarthome.co/home-automation/protocols/x10/. Accessed: Oct. 8, 2016.

[29] "What home automation protocol should I choose?," in Intellihome, 2015. [Online]. Available: https://www.intellihome.be/en/kbase/INSTEON/What_home_automation_protocol_should_I_choose_-2.html. Accessed: Oct. 8, 2016.

[30] "Introduction to Wi-Fi (802.11 or WiFi)," in CCM Benchmark, CCM, 2016. [Online]. Available: http://ccm.net/contents/802-introduction-to-wi-fi-802-11-or-wifi. Accessed: Oct. 8, 2016.

[31] E. Contributor, "Home Automation Protocols: A Round-Up," in Electronic House, Electronic House, 2016. [Online]. Available: https://www.electronichouse.com/smart-home/home-automation-protocols-what-technology-is-right-for-you/. Accessed: Oct. 8, 2016.

[32] Jim, "Bluetooth basics," in sparkfun. [Online]. Available: https://learn.sparkfun.com/tutorials/bluetooth-basics/common-versions. Accessed: Oct. 9, 2016.

[33] "Data rates using BLE," in Anaren atmosphere. [Online]. Available: https://atmosphere.anaren.com/wiki/Data_rates_using_BLE. Accessed: Oct. 9, 2016.

[34] "Bluetooth low energy," in CSR. [Online]. Available: https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=227336. Accessed: Oct. 9, 2016.

[54] "User login," 2014. [Online]. Available: http://www.engineersgarage.com/microcontroller/8051projects/LCD-digital-alarm-clock-AT89C51-circuit. Accessed: Oct. 29, 2016.

[55] "Arduino alarm clock," Instructables.com. [Online]. Available: http://www.instructables.com/id/Arduino-alarm-clock. Accessed: Oct. 29, 2016.

[56] "Arduino based digital clock with alarm," 2015. [Online]. Available: http://circuitdigest.com/microcontroller-projects/arduino-alarm-clock. Accessed: Oct. 29, 2016.

[57] "Tweet-a-Pot: Twitter enabled coffee pot," Instructables.com, 1932. [Online]. Available: http://www.instructables.com/id/Tweet-a-Pot-Twitter-Enabled-Coffee-Pot. Accessed: Oct. 29, 2016.

[58] Adafruit Industries, "PIR (motion) sensor ID: 189 - $9.95: Adafruit industries, unique & fun DIY electronics and kits,". [Online]. Available: https://www.adafruit.com/product/189. Accessed: Oct. 29, 2016.

[59] D.-L. Corporation and D.-L. Systems, "Wi-Fi motion sensor," 2012. [Online]. Available: http://ca.dlink.com/products/connected-home/wi-fi-motion-sensor/. Accessed: Oct. 29, 2016.

[60] SmartThings, SmartThings Shop, 2016. [Online]. Available: https://shop.smartthings.com/#!/products/samsung-smartthings-motion-sensor. Accessed: Oct. 29, 2016.

[61] Belkin International, "WeMo switch + motion," Belkin, 2016. [Online]. Available: http://www.belkin.com/au/p/P-F5Z0340-APL/. Accessed: Oct. 29, 2016.

[62] "Philips hue API," 2014. [Online]. Available: http://www.developers.meethue.com/documentation/how-hue-works. Accessed: Oct. 29, 2016.

[63] "Lutron Integration Protocol". [Online]. Available: https://www.lutron.com/technicaldocumentlibrary/040249.pdf. Accessed: Oct. 29, 2016.

[64] "Z-Wave plug-in smart dimmer - Z-Wave products," 2006. [Online]. Available: http://www.zwaveproducts.com/shop/brands/ge/z-wave-plug-in-smart-dimmer-1. Accessed: Oct. 29, 2016.

[65] SMARTHOME®, "Everspring ST815 Z-Wave wireless illumination sensor with LCD screen," 1995. [Online]. Available: http://www.smarthome.com/everspring-st815-z-wave-wireless-illumination-sensor-with-lcd-screen.html. Accessed: Oct. 29, 2016.

[66] SMARTHOME®, "HomeSeer HSM200 Z-Wave multi-sensor, gen 5," 1995. [Online]. Available: http://www.smarthome.com/homeseer-hsm200-z-wave-multi-sensor.html. Accessed: Oct. 29, 2016. [66] Home, "Training &Education," in Member Press Releases, Z-Wave Alliance, 2015. [Online]. Available: http://z-wavealliance.org/sigma-designs-introduces-z-wave-iot-ready-development-kits/. Accessed: Dec. 4, 2016.

[67] OpenZWave, "OpenZWave/open-zwave," GitHub, 2016. [Online]. Available: https://github.com/OpenZWave/open-zwave. Accessed: Dec. 4, 2016.

[68] one-fifth more, "Training &Education," Z-Wave Alliance, 2016. [Online]. Available: http://z-wavealliance.org/. Accessed: Dec. 4, 2016.

---

1. "Overview of the Internet of things," in "Telecommunication Standardization Sector of ITU," ITU, Jun. 15, 2012. [Online]. Available: http://handle.itu.int/11.1002/1000/11559. Accessed: Jan. 31, 2017.↩

2. A. J. Brush, B. Lee, R. Mahajan, S. Agarwal, S. Saroiu, and C. Dixon, "Home automation in the wild: Challenges and opportunities - Microsoft research," Microsoft Research, 2011. [Online]. Available: https://www.microsoft.com/en-us/research/publication/home-automation-in-the-wild-challenges-and-opportunities/. Accessed: Jan. 31, 2017.↩

3. "Home," in Insteon, Insteon, 2016. [Online]. Available: http://www.insteon.com/. Accessed: Oct. 6, 2016.↩

4. "Insteon Hub" in Insteon, Insteon. [Online]. Available: http://www.insteon.com/insteon-hub. Accessed: Feb. 1, 2017.↩

5. "Insteon Hub" in Insteon, Insteon. [Online]. Available: http://www.insteon.com/insteon-hub. Accessed: Feb. 1, 2017.↩

6. "Insteon Hub" in Insteon, Insteon. [Online]. Available: http://www.insteon.com/insteon-hub. Accessed: Feb. 1, 2017.↩

7. "WHITEPAPER: The Details,". [Online]. Available: http://cache.insteon.com/documentation/insteon_details.pdf. Accessed: Oct. 6, 2016.↩

8. "WHITEPAPER: Compared,". [Online]. Available: http://cache.insteon.com/documentation/insteon_compared.pdf. Accessed: Oct. 6, 2016.↩

9. "WHITEPAPER: Compared,". [Online]. Available: http://cache.insteon.com/documentation/insteon_compared.pdf. Accessed: Oct. 6, 2016.↩

10. "WHITEPAPER: The Details,". [Online]. Available: http://cache.insteon.com/documentation/insteon_details.pdf. Accessed: Oct. 6, 2016.↩

11. "WHITEPAPER: The Details,". [Online]. Available: http://cache.insteon.com/documentation/insteon_details.pdf. Accessed: Oct. 6, 2016.↩

12. "WHITEPAPER: The Details,". [Online]. Available: http://cache.insteon.com/documentation/insteon_details.pdf. Accessed: Oct. 6, 2016.↩

13. "Insteon API · Apiary,". [Online]. Available: http://docs.insteon.apiary.io/. Accessed: Oct. 6, 2016.↩

14. "Insteon API · Apiary,". [Online]. Available: http://docs.insteon.apiary.io/. Accessed: Oct. 6, 2016.↩

15. "Technology," in Insteon, Insteon, 2016. [Online]. Available: http://www.insteon.com/technology/. Accessed: Jan. 31, 2017.↩

16. "Insteon Connects," in Insteon, Insteon. [Online]. Available: http://www.insteon.com/connects. Accessed: Jan. 31, 2017.↩

17. "Wink | About Us". [Online]. Available https://wink.com/about/. Accessed: Feb. 1, 2017.↩

18. "Wink | About Us". [Online]. Available https://wink.com/about/. Accessed: Feb. 1, 2017.↩

19. "Wink | About Us". [Online]. Available https://wink.com/about/. Accessed: Feb. 1, 2017.↩

20. "Wink | Products". [Online]. Available https://wink.com/products/. Accessed: Feb. 1, 2017.↩

21. "Wink | About Us". [Online]. Available https://wink.com/about/. Accessed: Feb. 1, 2017.↩

22. "Wink | About Us". [Online]. Available https://wink.com/about/. Accessed: Feb. 1, 2017.↩

23. "A simpler smart home," Wink, 2016. [Online]. Available: http://www.wink.com/help/faq/. Accessed: Oct. 8, 2016.↩

24. "A simpler smart home," Wink, 2016. [Online]. Available: http://www.wink.com/help/faq/. Accessed: Oct. 8, 2016.↩

25. "A simpler smart home," Wink, 2016. [Online]. Available: http://www.wink.com/help/faq/. Accessed: Oct. 8, 2016.↩

26. "Wink | Products". [Online]. Available https://wink.com/products/. Accessed: Feb. 1, 2017.↩

27. "Wink FAQ - Wink@Home Wiki," 2015. [Online]. Available: http://wiki.winkathome.net/Wink_FAQ. Accessed: Oct. 6, 2016.↩

28. "Wink API · Apiary,". [Online]. Available: http://docs.winkapiv2.apiary.io/. Accessed: Feb. 1, 2016.↩

29. "ArduinoBoardUno," in Arduino, 2016. [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardUno. Accessed: Oct. 6, 2016.↩

30. "ArduinoBoardUno," in Arduino, 2016. [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardUno. Accessed: Oct. 6, 2016.↩

31. "ArduinoBoardUno," in Arduino, 2016. [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardUno. Accessed: Oct. 6, 2016.↩

32. "ArduinoBoardUno," in Arduino, 2016. [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardUno. Accessed: Oct. 6, 2016.↩

33. "ArduinoBoardUno," in Arduino, 2016. [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardUno. Accessed: Oct. 6, 2016.↩

34. "ArduinoBoardUno," in Arduino, 2016. [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardUno. Accessed: Oct. 6, 2016.↩

35. "ArduinoBoardPro," in Arduino, 2016. [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardPro. Accessed: Oct. 6, 2016.↩

36. "ArduinoBoardPro," in Arduino, 2016. [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardPro. Accessed: Oct. 6, 2016.↩

37. "ArduinoBoardMicro," in Arduino, 2016. [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardMicro. Accessed: Oct. 6, 2016.↩

38. "ArduinoBoardMicro," in Arduino, 2016. [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardMicro. Accessed: Oct. 6, 2016.↩

39. "ArduinoBoardMicro," in Arduino, 2016. [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardMicro. Accessed: Oct. 6, 2016.↩

40. "Raspberry pi Zero," Raspberry Pi. [Online]. Available: https://www.raspberrypi.org/products/pi-zero/. Accessed: Oct. 10, 2016.↩

41. "Raspberry Pi Zero,". [Online]. Available: https://shop.pimoroni.com/products/raspberry-pi-zero. Accessed: Oct. 10, 2016.↩

42. "Raspberry pi 1 model A+," Raspberry Pi. [Online]. Available: https://www.raspberrypi.org/products/model-a-plus/. Accessed: Oct. 10, 2016.↩

43. J. Adams, "Raspberry Pi Model B+," Mar. 07, 2014. [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/mechanical/Raspberry-Pi-B-Plus-V1.2-Mechanical-Drawing.pdf. Accessed: Oct. 10, 2016.↩

44. "Raspberry pi 2 model B," Raspberry Pi. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/. Accessed: Oct. 10, 2016.↩

45. J. Adams, "Raspberry Pi Model B+," Mar. 07, 2014. [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/mechanical/Raspberry-Pi-B-Plus-V1.2-Mechanical-Drawing.pdf. Accessed: Oct. 10, 2016.↩

46. "GPIO - raspberry pi documentation,". [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/gpio/README.md. Accessed: Oct. 10, 2016.↩

47. J. Adams, "Raspberry Pi 3 Model B," Jun. 10, 2015. [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/mechanical/RPI-3B-V1_2.pdf. Accessed: Oct. 10, 2016.↩

48. "Raspbian - raspberry pi documentation,". [Online]. Available: https://www.raspberrypi.org/documentation/raspbian/. Accessed: Oct. 10, 2016.↩

49. "Bone-original,". [Online]. Available: http://beagleboard.org/bone-original. Accessed: Oct. 10, 2016.↩

50. "BeagleBone Schematic," Jun. 28, 2012. [Online]. Available: https://github.com/CircuitCo/BeagleBone-RevA6/blob/master/BEAGLEBONE_REV_A6A.pdf?raw=true. Accessed: Oct. 10, 2016.↩

51. "Bone-original,". [Online]. Available: http://beagleboard.org/bone-original. Accessed: Oct. 10, 2016.↩

52. "BeagleBone Black,". [Online]. Available: http://beagleboard.org/black. Accessed: Oct. 10, 2016.↩

53. "Beagleboard: BeagleBoneBlack,". [Online]. Available: http://elinux.org/Beagleboard:BeagleBoneBlack. Accessed: Oct. 10, 2016.↩

54. "BeagleBone Green,". [Online]. Available: http://beagleboard.org/green. Accessed: Oct. 10, 2016.↩