End-User Programming for Home Automation

Samuel Lichtenberg
samlichten@gmail.com
GSET

Emily Zhang
em.c2.zhang@gmail.com
GSET

Matthew Continisio
mrcontinisio@comcast.net
GSET

Amanda Rumsey
amandarumsey@ymail.com
GSET

Raheem Scott Griffith
raheem.griffith@gmail.com
GSET

Abstract

Home appliances are, at present, set up in a manner that is unintuitive and convoluted. For example, to set the time on all devices, one must go around his house and individually, through button presses, interact with each single device. Coordinating functions between multiple devices requires even more menial button-pressing, and, in some cases, is wholly impossible. Currently, there is simply no good way for a user to quickly and easily set multiple devices to interact with each other.

To that end, we set out to create a new system of home automation. We sought usability, flexibility, the creation of a universal programming methodology, and cost-effectiveness. In short, we wanted to bring our system to the average user—not to only the wealthy or the tech-savvy.

In our paper, we explain, in detail, the multiple components involved in our system. The user interacts with a graphical programming language called BYOB, or Build Your Own Blocks, an extension of Scratch. Through this interface, the user has direct control over his individual home appliances as well as their interactions with each other. The user input in BYOB is transferred first to another programming language, Processing, which interprets that input and sends it to the Arduino microcontroller. Pre-loaded software on the microcontroller uses this information to control any devices that are connected to the system. To test the feasibility of our system, we reverse-engineered four devices: a desk lamp, a fan, a coffee maker, and an alarm clock. In our paper, we describe this process.

We conclude with an analysis of how our system meets the four original goals we set. In addition, we provide recommendations for directions in which future research could go, including improvements to the efficiency of our system, new and advanced features, and a full-scale user study.

1 Introduction

Imagine your home of the future: it is nine o’clock on a Saturday morning and you groggily awake to the smooth sounds of Miles Davis, your favorite jazz artist. You usually get up at seven, but, since it’s the weekend, your alarm clock knows to wait an extra two hours. You hit snooze a few times—it was a rough week and you’re tired—then finally shut off your alarm. The instant your feet hit the door, lights illuminate the hallway that leads you to your shower, which has already heated up by the time you reach the bathroom. Simultaneously, the coffee maker in your kitchen on the floor below starts brewing the exotic blend of Colombian coffee that you save for the weekends.

This dream scenario is, to most, far from reality. But we envision just such a home, controlled through a central point: a personal computer. We have designed and prototyped a system of home automation to make this a reality.

We kept in mind four goals for our system:

- Usability
- Flexibility
- Generalization
- Cost-Effectiveness

1.1 Usability

Our system should be easy to use for anyone to use. A user who is not technically inclined should be able to control his appliances as easily as a computer programmer would. Usability also implies speed; using our system should be faster than manually setting individual appliances.

1.2 Flexibility

Our system should allow home appliances to perform more complex tasks than before. For example, your lights could turn on whenever your alarm clock
went off. Appliances should also be able to remember your specific functions, such as creating a special brew for you or playing a certain song as your alarm.

1.3 Generalization

Appliances in our system should be unified under a single programming paradigm, using the same user interface and language.

1.4 Cost-Effectiveness

Our system should be relatively inexpensive to create. The average person should be able to afford our home automation system.

2 Designing a Home Automation System

Our home automation system involves three distinct software components. They are BYOB, or Build Your Own Blocks, the user interface; Arduino, the "translator" from the user input to the physical electrical signals sent to appliances; and Processing, the "middleman" that connects the two. In this section, we describe their purposes, functions, and methods of communication with each other.

2.1 BYOB

A key component of any home automation system is the user interface: it must be friendly and easy to use. Scratch, a graphical programming language designed in 2007 by MIT, is just that. A programming language intended for non-programmers, Scratch uses colorful blocks of code, with familiar commands such as if statements and while loops. However, in Scratch, device-specific blocks can’t be created. In fact, no blocks can be, and only a limited range of pre-programmed functions is available. To solve this problem, we used, in lieu of Scratch, the alpha version of Build Your Own Blocks (BYOB), a modified edition of Scratch developed by Jens Monig and Brian Harvey [4].

For our purposes, the physical process of block creation, though something no user will be forced to contend with, is fairly straightforward. The "Make a Block" button in BYOB is clicked and the name of the block entered. From there, other pre-existing blocks can be dragged into the "Block Editor", where the block’s function is denied. When the new block is run in a program, every single block that has been added in the Block Editor under it will be executed.

![Example of code creation](image)

Figure 1: This is an example of code we create

Examples of custom blocks are the Turn On Coffeemaker and Is Weekend? blocks shown above. This code would check if the date matched a weekend; if it did, it would turn on the coffeemaker at 7:15 AM. This setup, which reads like ordinary English, is intuitive and comfortable to use.

All complexities behind the creation of blocks, however, are left to us. The user only interacts with the interface shown below.

In the main editor, located on the center left, users simply drag blocks to their appropriate places. The first section where the user can place blocks is in the DEVICES block, which is where the user puts the blocks that turn on and off devices and perform other device-specific tasks. The second section is in the TIME block, which is where the user puts blocks that set the time and/or date. This creation of such a pre-existing skeleton program, into which users can drag blocks, was done specifically to facilitate the rapid construction of simple and effective programs. Once the user has finished, he must click the green flag in the upper-right corner, which automatically sets the time and date, and allows him to select one of the two buttons located beneath it, in the "stage". Clicking the Set Time button runs the blocks that are placed in the TIME block. Clicking the Program Devices button runs the blocks that are placed in the DEVICES block in a forever loop, which constantly checks to see if the blocks that the user placed should be executed or not. BYOB knows which section to execute based on the cursor's coordinates in the stage—if the cursor is within either the coordinates of the "Program Devices" or "Set Time" button and is pressed down, that specific code section will immediately run. Also located in the stage are images of all the devices currently connected to the system and their respective power states.
Each device in BYOB has a corresponding variable that keeps track of its power state. The variable names and values are all kept as strings of characters; the variable name precedes the value. For instance, a coffee maker in BYOB might have a corresponding variable CoffeePower, which would store its power state as either "on" or "off", in a Boolean fashion. The time in BYOB, too, is maintained by a string rather than numerical values. For BYOB to physically alter the devices themselves, however, it must first be sent to Processing.

### 2.2 Processing

Processing is a Java-based programming language developed in 2001 by Ben Fry and Casey Reas [6]. Its capabilities for communication between multiple programs that run simultaneously made it the perfect choice for our home automation system.

Processing receives the state of all active devices as well as the time, both located in BYOB, by first creating an object of the client class. This class has fields for IP address and port number—the IP address corresponds to whatever computer the user runs BYOB from, which enables him to run BYOB on his personal computer far from home and update his devices. The port number specifies which program Processing is to read data from. Port 42001 is the port number associated with BYOB.

As previously mentioned, BYOB stores its information in the form of strings. However, the Arduino software we use can only read information in the form of single characters. This is why Processing is so vital. Processing uses the readString() method of the instance of the client class to access the state of BYOB’s devices and time, which are updated continuously as long as the program remains active. It utilizes the charAt() method of the input string to search for characters at specific numerical positions in the BYOB strings, and then uses these characters to identify which variable is being altered and in what way. For example, a function programmed to check for modifications to the power state of the coffee maker, corresponding to, as previously mentioned, the string "CoffeePower", would first attempt to locate the character ‘C’ in the position where it would appear in the input string. If that character was found, the function would then locate a second character, ‘n’ or ‘f’, to determine whether the coffee maker was "on" or "off". If not, the string would be re-checked for different characters that correspond to other variables.

Once Processing has identified which variable is being altered and how, it stores it in variables defined in Processing: variables of type char (short for 'character') for the devices' power states, and time[], an array of character that stores the current time, including AM/PM. These values are all sent to the
Arduino software by writing to the serial port, declared in Processing as an object of the Serial class, part of the imported Serial library. This process of checking BYOB's strings and sending them to Processing is repeated until either BYOB or the Arduino software/hardware become unavailable, or the program is terminated.

Figure 3: Processing (A Java-based programming language)

2.3 Arduino

Arduino has its roots in Italy, where it was created by Massimo Banzi as a response to the lack of powerful, cheap microcontrollers to run projects [7].

The term Arduino actually refers to two parts of the Arduino: the tangible hardware portion (referred to as Arduino microcontroller) and the software (referred to as Arduino software), where programs, called sketches, can be written and uploaded to the Arduino microcontroller to be executed.

The Arduino microcontroller is essentially a small computer, connected to the user's computer via a USB, or Universal Serial Bus, cable. The USB cable acts as a virtual serial port and is how data, in the form of single characters, is sent from Processing to Arduino.

The Arduino software is based on the Processing language, and is uploaded once to the Arduino microcontroller's flash memory, where it is stored indefinitely. This software reads from the serial port and writes directly to the Arduino microcontroller based on which variables and values are sent. The variables that represent each device's power state are sent from Processing in a specific order, and this order is used by the Arduino software to determine which device is being altered. Each of these variables is sent to the serial port every single time the loop in Processing runs, even if they have not changed, to ensure that the order is preserved.

From this point onward, what instructions the Arduino software gives to the microcontroller are entirely dependent on which devices are altered. If a time on an LCD screen is being updated, the Arduino software would utilize the SerialWrite() function; if a device is being turned on or off, the digitalWrite() function would be used to send either 5 or 0 volts to a specified pin on the Arduino microcontroller. The latter function has two parameters: pin number and amount of voltage sent. The Arduino has the capability to send a voltage between 0 and 5 using the analogWrite() function, but this is unnecessary for our system.

Figure 4: The Arduino microcontroller

3 Prototyping a Home Automation System

To test the feasibility of our design, we reverse-engineered four devices and connected them to BYOB in the manner previously outlined. These
devices were a coffee maker, a desk lamp, an alarm clock, and a fan. Here, we will outline our methodology.

3.1 Arduino Microcontroller

We used an Arduino Mega microcontroller to segue from the software design portion of our system into the tangible hardware prototypes.

The Mega has 54 electrical pins that can be used as either input pins that read information from sensors, or as output pins that send out a voltage as dictated by the program uploaded to the board. To these pins, we connected wires that connected to and controlled our devices. Each device has different parts that connect in different ways, and, in the following, we will explain how each device was specifically reverse-engineered and connected to the Arduino.

3.2 Desk Lamp and Fan

The desk lamp and fan were similar in that they each had only two states: on and off. To reverse-engineer them, we disabled the switches so they would always be in the on position when plugged in. Then, taking the power cord, we cut the hot wire and connected both ends to two pins on a relay, while leaving the neutral wire alone. Relays are simply electrical switches.

![Figure 7: Relay](image)

Figure 7: Relay [9] - There are two sides to a relay: the load, and the input, each of which has two pins. The Arduino connects to one input pin, and the device being controlled connects one load pin. If there is voltage owing to that input pin from the Arduino, the relay will connect the device to the power outlet, which the other half of the relay is connected to, and power will come through. If not, the device will remain off. The other input pin on the relay is connected to a ground pin on the Arduino.

On the other end of the relay, we attached two wires, one of which led into the ground pin on the Arduino microcontroller, the other of which led to the 5v pin on the microcontroller. Thus, we created our own digital switch and removed the need for a manual switch. Whenever the 5v pin sent out 5v, it would be mimicking a switch in the on position and electricity would flow through the wires; whenever the 5v pin sent out 0v, it would be mimicking a switch in off position and electricity would not flow.

3.3 Coffee Maker

The coffee maker was the first device we attempted to reverse-engineer and proved to be the most difficult. Within itself, the coffee maker is actually comprised of two electrical components: the heating element and the LCD.

3.3.1 Heating Element

We reverse engineered the heating element similarly to the way we reverse engineered the desk lamp and fan. We cut the hot wire that lead to the heating element and attached the split wires to two ends of a relay. On the other two ends of the relay, we attached wires that led to a ground and a 5v pin on the Arduino that could control the state of the heating element.

3.3.2 LCD

Initially, we tried to figure out how to produce times on the built-in LCD screen by writing a program to cycle through all of the possible pin configurations. However, we found that each pin did not control individual segments on the seven-segment displays; in order to control the LCD effect, we would need to use a quartz crystal oscillator. We decided to re-place the coffee makers LCD with our own, more easily programmable 16x2 LCD, which we could more easily program.

![Figure 8: Desk Lamp](image)

Figure 8: Desk Lamp

![Figure 9: Fan](image)

Figure 9: Fan

![Figure 10: The hardware behind Mr. Coffee](image)
3.4 Alarm Clock

We aimed to reverse-engineer the alarm clocks two main parts the alarm and the clock.

3.4.1 Clock

For the clock portion, we decided to use our own set of seven segment displays because they were easier to program than the LCD that came with the clock. Each segment of the display was connected to a pin on the Arduino and was individually controlled. Displaying numbers and times on the LCD required activating certain combinations of segments and the use of a script completely different from the one used to run the 16x2 LCD script. Time in both types of displays were synchronized and controlled by BYOB.

3.4.2 Alarm

The alarm was simple to control. One of the two wires leading from the alarm speaker was connected to a ground pin on the Arduino, and the other was connected to a 5v pin, creating another electronic switch that could turn the speaker on and off.

4 Conclusions

After successfully designing and prototyping our home automation system, we must examine if the goals we initially outlined were successfully met.

4.1 Usability

Our system is usable. Instead of programming devices in a low level language, users interact with a graphic, intuitive interface. No prior programming experience is necessary to understand how the BYOB blocks function. In addition, our system makes programming devices and times simultaneously a much simpler process. While it may be easier to set some individual devices manually, such as a desk lamp, when considering the fact that in an average home there are likely a great number of devices that need to be set, it is much simpler and easier for a user to interact with his appliances from one central point.

4.2 Flexibility

Our system is flexible. There are tasks our system could do easily that would be impossible manually. There is no manual way to program event-driven tasks—for example, to have the coffee maker turn on only when the alarm clock goes on. With BYOB, this is simple.

4.3 Generalization

Our system has a universal platform. In using BYOB as a central interface that every device interacts with, we ensure that the programming methodology for each device is nearly identical. The process a user employs to set a coffee maker would apply to how he sets his lights, fan, and alarm clock.

4.4 Cost-Effectiveness

Our system is cost-effective. Because we use only open-source software, our code is free and is
accessible by anyone. The only cost in our system, at about 65 dollars, is an Arduino mega, which can be used to set every device in a home.

4 Recommendations for Future Work

There are several interesting directions for future research that we did not have time to fully explore in our project.

5.1 More Types of User Input

There is the possibility of more efficient and sophisticated types of user input. Rather than interfacing directly with BYOB, a user could employ a voice-recognition system. We experimented with the possibility of Sphinx, an open-source voice-recognition software developed by students at Carnegie Mellon University, but did not make tangible progress. There are other possibilities as well; future systems could employ even more sophisticated types of recognition, such as mood or body-temperature recognition.

5.2 User Study

To verify whether or not our system is truly as usable and flexible as we believe it to be, future work should include a large-scale user study to assess how comfortable the average person feels using BYOB and how much they can accomplish with it.

5.3 Improvements on BYOB

There are also ways that future work could improve upon restrictions we encountered in our system. For instance, while BYOB has many advantages, there are some disadvantages as well. We were not able to automatically synchronize time between BYOB and the computer BYOB is run on; the time needs to be manually set by the user every time BYOB is launched. This is a clear limitation. Furthermore, it is difficult for our BYOB setup to perform certain specialized functions. While BYOB is adept at managing everyday commands, such as instructing your coffee maker to turn on at 9 AM every morning, it does not handle unorthodox functions well. This would include a command such as setting a fan to turn on for 5 seconds, off for 5 seconds, on for 10 seconds, and off for 10 seconds. Finally, if a user wishes to add a new device to an existing home automation system, every single step of our system must be individually modified to accommodate the new device, an inefficient process.

6 Final Thoughts

Designing and prototyping a home automation system in less than four weeks was no easy task. We essentially had to learn three relatively new programming languages and concoct a method for communicating between all of them; a task for which no tutorials were present and any information was scarce. We had to reverse-engineering devices for which we had neither schematics nor documentation. In doing so, we frequently ran into problems, and were often forced to restart entirely from scratch. But despite these setbacks, we believe that we have laid the groundwork for future research in the field of home automation. All the software we used is open-source, and we wholly encourage others to build upon our ideas. We hope that others can use our work as a guide in creating a usable, flexible, universal, and cost-effective home automation system.

7 Acknowledgements

We would like to thank Blase Ur and Michael Littman for being our mentors as well as the UPOD group at Rutgers University.
References


