Scratchable Devices: User-Friendly Programming for Household Appliances

Jordan Ash, Monica Babes, Gal Cohen, Sameen Jalal, Sam Lichtenberg,
Michael Littman, VukosiMarivate, Phillip Quiza, Blase Ur, and Emily Zhang

Rutgers University, New Brunswick, NJ
jordash@eden.rutgers.edu, babes@cs.rutgers.edu,
galcohen@eden.rutgers.edu, samjalal@eden.rutgers.edu,
samlichten@gmail.com, mlittman@cs.rutgers.edu,
vukosi@cs.rutgers.edu,pquiza@eden.rutgers.edu,
blase@blaseur.com, 1800.ehz.hang@gmail.com

Although household devices and home appliances function more and more as network-connected computers, they don't provide programming interfaces for the average user. We first identify the programming primitives and control structures necessary for the universal programming of devices. We then propose a mapping between the features necessary for the programming of devices and the existing functionality of Scratch, an educational programming language we use as a basic interface between the devices and the users. Using this modified version of the Scratch language, we demonstrate usage cases in which novice programmers can program appliances, increasing their functionality and ability to be customized. We also show how standardizing this programming paradigm can facilitate knowledge transfer to new devices. We conclude by discussing our experiences prototyping programmable appliances.

Keywords: educational programming; end-user programming; home automation; household devices; programming languages; scratch; ubiquitous computing; usability;

1   Introduction

From the Jetsons' futuristic home to the personalized temperature, music, and lighting systems in Bill Gates' estate [1], the concept of home automation intrigues and fascinates the average person. From a hardware perspective, home automation is already a reality as equipment for interconnecting devices is commercially available. With the arrival of an 'Internet of Things', a future proposed by major electronics manufacturers [2] in which every household device and appliance would have an IP address and be connected to the internet, capable hardware would become even more ubiquitous. However, the manner in which the average person might interact with and customize household devices has not been well studied.

One source of interest with home automation is that it is a mode of self-expression in addition to adding convenience and functionality to one's lifestyle. A home is important: it is, to many, a reflection of who they are and what they value. This human desire for customization propelled personalized ringtones into a multi-billion dollar industry [16]. It is a natural step to make a home's electronics and appliances into something personal, creative, and increasingly functional.

Existing home automation systems focus on the hardware for connecting devices, paying little attention to creating a usable and universal programming paradigm capable of controlling any type of appliance. The ability to fully customize a home requires the logical and control structures of an expressive programming language, whereas existing systems generally provide only a menu-based system for controlling appliances. Our goal is to enable users to write simple programs that add unique functionality, allow devices to communicate, and save time.

The overriding goal of our project is to enable the average person, somebody with no previous programming experience, to write programs for customizing their home appliances. To this end, we propose using a variant of the graphical programming language Scratch, designed for novice programmers [13], as the standard for programming home devices since Scratch is meant to be intuitive for first-time programmers.

In this paper, we lay out a comprehensive yet usable system for programming home devices. We outline the primitive operations necessary for such a programming system and show how the style and metaphors of the Scratch language directly support these operations. As initial support for our concept, we demonstrate usage cases building on these primitives. These compact example programs demonstrate novel, useful functionality for an automated home. We also discuss the construction of our prototype "Scratchable Devices."

2   Background and Related Work

We first explain the physical communication standards and programming methods of existing home automation systems and introduce the Scratch programming language.

2.1   Existing Home Automation Systems

Existing residential home automation systems primarily focus on communication protocols. X10 [17], one of the most widely used protocols, communicates over power lines. INSTEON, designed to provide added reliability over X10, enables communication over both power lines and wireless RF [14]. Protocols that are primarily wireless include Z-Wave [18]. Although these standards specify how home devices communicate, they do not prescribe usability standards or programming methodologies. Since our investigation focuses on the user interface, our proposal could be considered orthogonal to existing communication standards.

The control systems for existing home automation solutions range in complexity from touch screens to intricate (at times unintuitive) software packages that allow

users to write programs in scripting languages. However, each of these approaches treats usability and functionality as mutually exclusive, in contrast to our proposed Scratchable Devices.

Hardware methods for controlling household devices include universal remote controls and wall-mounted touch screens [9]. Touch screen systems generally allow users to control devices, monitor their state, and set a device to change states at some later time. Software for smart phones and tablets, such as the iPhone and iPad, is also available from multiple manufacturers [4], [5].

Computer software, either local or web-based [8], is also commonly used to control home automation systems. Often, this software is controlled through a series of menus that closely mirror the functionality of touch screen interfaces. The user's ability to write their own programs is often limited to writing macros, as in the Active Home program [17]. These macros lack the control structures of full programming languages, such as iteration and conditional statements.

Some home automation systems do grant limited access to control structures. For instance, Zeus [15] allows users to define conditional statements, although this language lacks iteration and other hallmarks of fully featured languages. The PowerHome program provides customers with a full set of commands in the scripting language of their choice [11], but these languages are designed neither for novice users nor for the specifics of home-automation applications. None of these approaches successfully merges functionality with usability.

2.2 Scratch

Scratch, a graphical and interactive programming environment developed to foster children's excitement and interest in computer programming, has a number of features that make it suitable for first time programmers [13]. Scratch makes programming simple by representing statements as drag-and-drop blocks that snap together. These blocks include basic elements such as variables, conditionals, loops, inputs, and Booleans. Scratch is object-centered, where each object is called a sprite. Each sprite appears as an image whose graphical appearance, termed a costume, is under user control. All sprites appear on a stage, which is an area of the screen that displays each sprite in its current costume.

Users write event-driven scripts to control sprites. For instance, a user could create one sprite that looks like a cat and another sprite that looks like a dog. After choosing the cat sprite by clicking on it, the user could write a script that changes the cat's costume whenever the spacebar is pressed or make the cat meow if the dog is too close. Sprites can also communicate with each other through a broadcast mechanism.

To make household devices programmable to users without prior programming experience, the language they use must be simple to learn, intuitive, and present a uniform interface across different household devices. Because of Scratch's gentle learning curve and visually attractive, non-intimidating interface, it is used with great success in schools nationwide, from elementary through high school, and even in some introductory computer science courses at universities [6], [13].

3   Programming Primitives for Home Automation

To design a general-purpose programming language for household devices, we first propose a set of language-independent programming primitives that formalize users' interaction with their appliances. At a high level, we propose that these primitives be object-centered since users see each household device as a separate object. We also suggest that the language standardize the appearance and functionality of outputs and inputs. The output of each object will be state changes. Inputs to our language consist of sensors (on devices or free-standing), physical interaction with individual devices (such as button presses), and state queries. We additionally identify inter-device communication as an essential primitive. Our primitives are outlined in **Fig. 1.**

3.1   Outputs As State Changes

From a programming perspective, the most important property of a household device is its current state. The simplest devices have binary state; for example, a lamp can be either 'on' or 'off'. More complicated devices may have many different states, but the user should still conceive of all manipulations to the device as a state change. For instance, a washing machine could have multiple modes of being 'on', such as being 'on in *Permanent Press* mode' or 'on in *Bright Colors* mode'.

   For both simple and complex devices, we propose that the primary output of a home automation programming language be changes of state on a per-device basis. Therefore, a simple device will have only a 'Turn On' block and a 'Turn Off' block. More complex devices would add a drop-down menu of possible modes to the 'Turn On' block. For instance, a washing machine's Turn On block could be edited via a drop-down menu to specify that the device must turn on in '*Permanent Press*' mode.

3.2   Inputs

Many sensor readings will be available globally to all objects, including the time and date as well as environmental conditions. Sensor readings related to time include blocks such as 'Time' and 'Is Workday', where the latter block returns a Boolean value. A number of global sensors could measure the environment, such as the 'Temperature' or 'Brightness' in a particular room, allowing users to create a 'Smart House'. These global sensors could be collocated with devices or placed independently throughout a home.

   Local inputs include both a user's physical interactions with a device and state queries. For instance, the action of a user pressing a button or turning a knob on a particular device would be represented, respectively, by a Boolean block and a block that returns a number. State queries, such as a 'Device Is On?' block and a washing machine's 'Device Is In [Cotton Cycle] State?' block, would return a Boolean value.
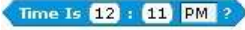
Novel Language Inputs	Language	Novel Language Outputs
Time	Iteration	Basic Device State Changes
`Time Is 12 : 11 PM ?`	Relational/ Logical Operators	`Turn On`
Environment Sensors	Conditionals	Complex Device State Changes
`Temperature (F)`	Delays/Timing	
Physical Interaction	Randomness	`Turn On In Permanent Press ... Mode`
`HR Button Pressed?`	Broadcast Communication	
State Queries	Multimedia - Audio	
`Device On?`		

Fig. 1. This table outlines the novel inputs and output primitives necessary for controlling household devices with a graphical language. Necessary control structures are also indicated.

3.3 Communication Between Devices

To permit rich interaction between household devices, a communication channel is necessary. Since the number of purpose-built communication channels between particular devices would increase exponentially with the number of devices, we propose that a user send messages as broadcasts into a cloud, with low-level communication transparent to the user. Each device would have an input block for receiving messages. Thus, a toaster could send the message 'Turn On Coffee Maker', and a coffee maker would have a corresponding script written by the user that listens for the broadcast 'Turn On Coffee Maker' and responds appropriately. Since messages are broadcasts, our proposed programming language enables one message to simultaneously communicate with many devices. Thus, the message 'Breakfast Time' could coordinate multiple devices working together every morning.

3.4 Control structures

To present a user with the power to fully express his or her ideas, our language must contain the core control structures of modern programming languages. Iteration, conditionals, and both relational and logical operators are all essential. In addition, pseudo-randomness, multimedia functions, and the ability for objects to communicate all greatly increase the usefulness and expressiveness of this programming language.

3.5 A Device's Logic

A Scratchable Device should be sold with its logic and functionality programmed in our language and visible to the user. An interested user could view and modify the core operations of a device. For instance, a user could reprogram the buttons of a

coffee maker to reflect the way he or she uses the device. Once the user writes a program, this program could reside on a centralized controller or on the device itself.

However, the logic that ensures the safety of a device should reside on the device itself and should operate independently of the programmable functions. For instance, a device with a heating element should be able to turn itself off based on time or temperature thresholds specified by the manufacturer in order to prevent a fire, regardless of whether a user's program is requesting the device to remain on.

4   Mapping Primitives to Scratch and Prototyping

The programming primitives we have outlined for Scratchable Devices can be mapped to the graphical programming language Scratch. Each device in a home automation system can be considered a sprite in Scratch, which functions as an object. The state of the device can be represented visually by a Scratch costume, and a user can view the current state of all the devices on the Scratch stage.

We have prototyped our programming paradigm using BYOB (Build Your Own Blocks) [10], a variant of Scratch in which we can create customized programming blocks. These blocks follow the style and structure of Scratch, with blocks such as 'Turn On' that appear as if they were native to Scratch.

5   Usage Cases

We believe that the ability to program home appliances will enable users to do *more*, extending the capabilities of these appliances beyond what is typical of devices today. In addition, users should be able to accomplish tasks *faster* by using programming rather than menus, and learn to use new devices *sooner* because of the unified interface across machines. To demonstrate the functionality of our proposed programming paradigm, we present the following usage examples.

5.1   'Clapper' for a Light

Fig. 2. This script allows any device to be turned on or off by clapping.

Our first usage case illustrates how novel yet useful features can be added to a device with short programs. In **Fig. 2**, we present a script that uses the computer's

microphone and Scratch's'loud?' Boolean, true when the microphone's input exceeds a threshold, to turn on or off a lamp. This code is placed as part of the lamp's sprite.

Knowing how to program a 'Clapper' light makes it very easy to make a 'Clapper' fan or alarm. In fact, the Scratch code is identical, but would instead be placed within the sprite for the fan or alarm clock, respectively. Reusability of code makes introducing a new device to a household very simple. In contrast to current devices, the learning curve for new Scratchable Devices would be minimal.

5.2   Wake Up With Coffee

The Scratchable Devices paradigm makes inter-device communication seamless, whereas this sort of communication is complex or impossible in current systems. In**Fig. 3(a)**, an alarm clock tells a coffee maker to turn on at 7:57 AM, waits three minutes, and sounds the alarm's buzzer. Thus, coffee will be ready when the user wakes up. The script in **Fig. 3(a),** written in the alarm clock's sprite in Scratch, broadcasts the message 'make coffee'. The script in **Fig. 3(b)** must be included in the coffee maker's sprite in order to define what 'make coffee' means.
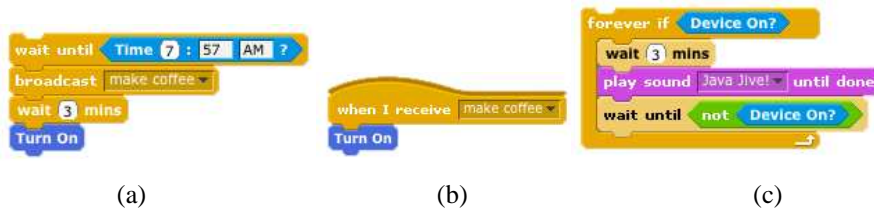

(a)   (b)   (c)

Fig. 3.(a) An alarm clock requests coffee and sounds its buzzer afterwards, waking the user to a fresh-brewed pot. **(b)** The coffee maker responds to "make coffee" by switching on.**(c)** Each time the coffee maker goes on, it announces when the coffee is done.

5.3   'Coffee is Ready' with a Song

Scratchable Devices can also be customized with rich multimedia, similar to cell phone ringtones. For instance, users can program their coffee maker to play their favorite song when their coffee is ready, as shown in **Fig. 3(c)**.

5.4   Activity Simulator for Vacations

Elements common to most programming languages can also add functionality to homes. **Fig. 4** depicts a program that uses a pseudorandom generator to turn a house light on and off randomly after 6 PM, deterring potential thieves while a user is away on vacation. Pseudorandomness, while simple to employ in a language like Scratch, would be difficult to implement without a fully-featured programming language.
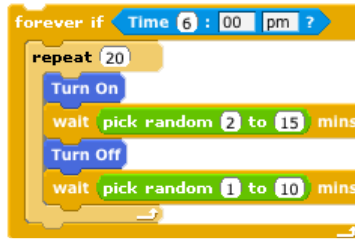
Fig. 4. A house light is turned on and off during evening hours to deter would-be robbers.

5.5   Setting Alarm Time

In existing systems, changing a clock's alarm time from 9:00am to 8:45am would generally require around 68 button presses to advance both the minute and hour. However, small changes in Scratchable Devices can be effected by editing a text box, or the user can reprogram existing buttons for common operations. This same mechanism works across devices, resulting in knowledge transfer.

More usage examples can be found on our scratchabledevices.com website.

6   Prototype Construction

We have constructed prototype Scratchable Devices, including lights, fans, and alarm clocks. While our prototypes demonstrate the feasibility of this programming paradigm and provide an upper bound on the hardware costs, existing or future hardware systems for home automation could just as well form the hardware layer of Scratchable Devices. Homemade Scratchable Devices could similarly exist. The primary contribution of this paper is the *user experience* in the household devices domain rather than a specification for hardware-level control.
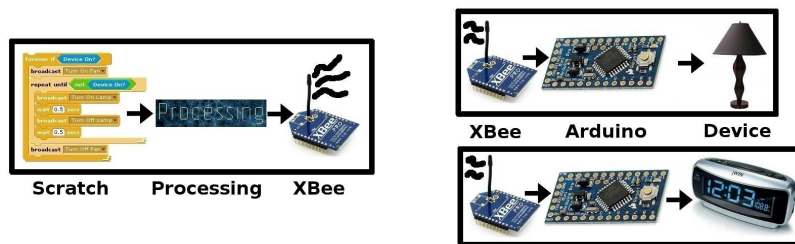

Fig. 5. Diagram of our prototype system, which uses the Processing language to pass commands from Scratch wirelessly (using XBee) to devices controlled by microcontrollers.

On the user experience level, our Scratchable Devices prototypes have been implemented using the BYOB (Build Your Own Blocks) variant of Scratch. We chose BYOB because a number of our proposed programming primitives, including

the `Turn On' block for each device and our use of time as a global sensor, required new types of Scratch blocks.

To allow BYOB to communicate with physical devices, we have used the Processing language as a translator [12]. A program in the Processing language runs in the background at the same time as Scratch, detecting when a variable in Scratch has been changed. Each Scratch sprite (household device) has one or more variables indicating its state. Each physical device has a globally unique 3-byte ID, and each corresponding sprite's variables begin with that ID. Our program in Processing takes messages received from Scratch and broadcasts them without modification over an XBee 802.15.4 wireless module.

Every Scratchable Device, such as a lamp or fan, also has an XBee 802.15.4 wireless module connected to an Arduino hobbyist microcontroller [3]. Since communication takes place on a broadcast medium, all devices receive all messages. In a commercial implementation, it would be wise to define a bootstrapping procedure in which newly purchased household devices inform the local Scratch instance of their device ID and feature set. That device would automatically appear as a sprite (object) in the Scratch instance.

When a Scratchable Device receives a message addressed to its ID or a broadcast message, such as 'Breakfast Time', it is programmed to read the message payload and change the state of the device accordingly.We have reverse-engineered a number of household appliances and inserted relays, LCD screens, and additional mechanisms that are controlled by Arduino microcontrollers. We have also attached small circuits to the buttons of the devices so that the microcontrollers can detect physical button presses and pass this information to Scratch, via Processing.

7   Conclusion and Future Work

We presented Scratchable Devices, a programming paradigm for household devices that allows the average person to program his or her appliances using a graphical programming language. The logical and control structures of a programming language provide additional functionality to household devices and potentially simplify existing tasks. We presented the programming primitives necessary for interaction with household devices and showed how the graphical, educational programming language Scratch maps directly to these primitives. We provided a short list of simple usage cases in our programming paradigm to support our claims.

In the near future, we will conduct user studies to verify the usability of the system compared to standard interfaces. Using this kind of formative design-evaluation [7], our interface will be made entirely with the average user in mind.We are particularly interested in verifying that users with no programming experience can, with minimum time and effort, learn to program devices in ways that add novel operations. We are also interested in any cognitive dissonance between our paradigm and the conception the users have of their household devices.

In the long run, we want to encourage electronics manufacturers to employ a version of our programming paradigm as an industry standard. Both similar and

dissimilar devices should all use the same programming protocol. This consistency throughout the home would make for a simplified, richer user experience.

Acknowledgments

References

1. Allon, F.: Space, Media Flows and 'smart' living in the Absolute Present. MediaSpace: place, scale, and culture in a media age, (2004) 253
2. Vance, A.: You Too can Join the Internet of Things, http://bits.Blogs.Nytimes.com/2010/09/20/you-Too-can-Join-the-Internet-of-Things/.
3. Banzi, M.: Getting started with arduino. Make Books (2008)
4. Control 4: My Home - iPad, http://www.control4.com/residential/products/mobile/myhome-Ipad/.
5. Creston Electronics Inc.: Apple & Creston. iPhone, iPod and iPad Mobile Control, http://www.Crestron.com/solutions/apple_mac_iphone_ipod_ipad_control/.
6. DesJardins, M., & Littman, M.: Broadening Student Enthusiasm for Computer Science with a Great Insights Course. (2010) 157-161
7. Egan, D. E., Remde, J. R., Gomez, L. M. et al.: Formative Design Evaluation of Superbook. ACM Transactions on Information Systems (TOIS), **7** (1989) 30-57
8. Hawking Technologies Inc: HomeRemote Pro Home Automation Starter Kit, http://www.Hawkingtech.Com/.
9. HomeSeer Technologies LLC: Home Automation Systems, http://store.Homeseer.com/store/Touchscreen-User-Interface-Plug-Ins-C98.Aspx.
10. Monig, J., & Harvey, B.: Build Your Own Blocks University of California at Berkeley. (2010)
11. PowerHome Automation LLC: PowerHome Automation, http://www.Hawkingtech.Com/.
12. Reas, C., & Fry, B.: Processing: A programming handbook for visual designers and artists. The MIT Press (2007)
13. Resnick, M., Maloney, J., Monroy-Hernández, A. et al.: Scratch: Programming for all. Commun ACM, **52** (2009) 60-67
14. SmartLabs® Inc: How INSTEON Works, Url: http://www.Insteon.net/about-Howitworks.Html.
15. Trax Softworks Inc.: Zeus Description and Features, http://www.Hawkingtech.Com/.
16. Wagner, A.: The Mobile Storefront: Let Your Fingers do the Shopping. (2005) 45
17. X10.com: Standard and Extended X10 Code Protocol, http://software.x10.com/pub/manuals/xtdcode.Pdf.
18. Z-Wave Alliance: About Z-Wave, http://www.z-Wave.com/modules/AboutZ-Wave/.