BEESM, a Block-based Educational Programming Tool for End Users

Mazyar Seraj
Institute of Computer Science, University of Bremen
Cyber-Physical Systems, German Research Center for Artificial Intelligence
28359 Bremen, Germany
seraj@uni-bremen.de, mazyar.seraj@dfki.de

Serge Autexier
Cyber-Physical Systems, German Research Center for Artificial Intelligence
28359 Bremen, Germany
serge.autexier@dfki.de

Jan Janssen
Cyber-Physical Systems, German Research Center for Artificial Intelligence
28359 Bremen, Germany
jan.janssen@dfki.de

ACM.
NordiCHI'18,, September 29-October 3, 2018, Oslo, Norway
ACM 978-1-4503-6437-9/18/09.
https://doi.org/10.1145/3240167.3240239

Abstract

Visual programming as a technique to support users to learn programming is an active field of research. Bringing together the hot topic of Smart Environments and the visual programming paradigm we present the *Block-based End-user programming tool for SMart Environments (BEESM)*. The dedicated application domain engages users to get interested in programming. *BEESM* allows to learn the general purpose of programming and rapidly prototype and customize applications in the context of smart environments. This approach enables users to program smart environments, microcontrollers and mobile robots one at a time and in combination with each other. It provides a block-based rapid programming tool as a hassle-free environment for educators and engineers to make it compatible with different smart devices and environments.

Author Keywords

Visual programming; end-user programming; smart environment; Google Blockly; user interface design.

ACM Classification Keywords

H.5.2 [User Interface]: Graphical user interfaces; D.2.6 [Programming Environments]: Graphical environment.

Introduction

Inexperienced users and novice programmers, the targeted users of *BEESM*, typically have difficulties to understand the requirements of designing, executing and debugging software programs [8, 10]. Learning and recalling code syntax is hard because it requires a high cognitive load for the targeted users. Furthermore, assembling and manipulating code structures is error prone for different reasons [2, 10]. Compared to the complexity of source code programming, visual programming has a great potential to facilitate programming for these users [8, 12]. In particular, block-based programming emerged as a form of visual programming that reduces the cognitive load by encapsulating the code into smaller code chunks and relying on recognition of blocks instead of remembering the code syntax. Furthermore, the blocks assist users to assemble the code without basic errors as manipulation of code structure [2, 10]. For example, a string block can be plugged into a length block, but not into a math arithmetic block. Nevertheless, inexperienced users who are interested in having an easy way to customize their programming ideas into real-world environments face two issues while working with block-based educational programming tools. First, the tools are not applied to real-world objects and environments. Second, although the tool can in principle be applied to the real and tangible objects, actually adapting it to, for instance, mobile robots, microcontrollers or smart environments requires to work and become familiar with many other tools [6, 7, 10, 11].

This paper contributes the design and development of *BEESM*, a *Block-based End-user programming tool for SMart Environments* with having both aforementioned issues in mind. This approach enables users to program smart environments, microcontrollers and mobile robots one at a time and in combination with each other. Having a tool like *BEESM* helps users to have a short time span between the development of ideas and their implementation in real environments. This should leverage the interest of users for programming and help them by adequate computational support.

State of the Art

This section introduces end-user programming techniques and design principles and gives a brief overview of visual programming in commercial and educational tools.

End-user programming (EUP) research aims at enabling inexperienced users and novice programmers to learn and make programs [7, 10, 13]. Regarding to the EUP systems, six learning barriers which are design, selection, coordination, use, understanding and information were addressed in [10]. The definition of these six barriers are based on the concept of programming interface which includes programming language constructs such as loops, conditionals, operators, variables, functions, objects and libraries. With this regard, EUP tools should be easy to learn, easy to use and helpful to minimize the distractions of users' goal and the learning barriers [9, 10]. Block-based commercial and educational programming tools for users resulted from applying a set of techniques and design principles for EUP from [8, 9, 10, 12, 13].

Behavioral programming is a technique in which all independent behaviors are wrapped into individual events which are executed at run-time [5]. However, its expressivity is limited as it does not, for instance, allow to program the change of a status of a smart object depending on the status of another object. Hence we disregarded this technique in our approach. Trigger-action programming is another type of end-user programming used to enable inexperienced users to program a smart home objects [6]. A prominent example is the trigger-action programming method included in the “if this then that” (IFTTT) service. IFTTT executes an action when an event occurs. Another form of end-user programing is programming by demonstration

(PBD). Using PBD in the context of a smart home, inexperienced users are able to demonstrate a situation and a desired behavior for smart devices using a set of examples [4]. These approaches allow users to apply changes to smart objects, but they do not help to learn general purposes of programming.

Visual programming
Visual or graphical programming was used to enable inexperienced users to understand and create programs with little training [3]. Visual programming environments are intended to reduce the complexity of programming for inexperienced users and novice programmers. Implemented in visual programming environments or editors such as Scratch and mblock, they can be used to create interactive applications (using Scratch) and to program robots/Arduino (using mblock) [1, 7]. Droplet is another block-based educational tool helping to work with both blocks and text code and allows users to become familiar with code syntax [1]. Visual programming, in particular Google Blockly, has been used in educational and commercial tools, such as UNC++Duino [11] and CustomPrograms [7].

Overview of BEESM

BEESM is designed as a rapid block-based programming tool for educational purposes to help inexperienced users and novice programmers to program. It provides visual programming in a web-based environment to program mobile robots, microcontrollers and smart environments. In this tool, in order to simplify programming for the users, *BEESM* supports to encapsulate behaviors of smart objects in different functions and to present them as basic programming primitives. *BEESM* can thus be used to program:

- *Smart Environments*: *BEESM* can be adapted and applied to smart environments if they support the web socket communication protocol and Remote Procedure Call (RPC) technology.
- *Mobile Robots*: *BEESM* can be applied to any mobile robot with autonomous navigation running on Robot Operating System (ROS).
- *Microcontrollers*: *BEESM* can be applied to any microcontroller running on Arduino Software by generating Arduino code for them.

BEESM is primarily designed for users to enable them to make programs. It includes different programming language features like variables, conditionals, loops, predefined functions and operators based on the visual editor Google Blockly. *BEESM* provides visual programming for the Hypertext Preprocessor (PHP) programming language and Arduino code. It allows to program smart environments and mobile robots using PHP. Furthermore, for Arduino programming, it takes advantage of blocklyDuino which is used in [11].

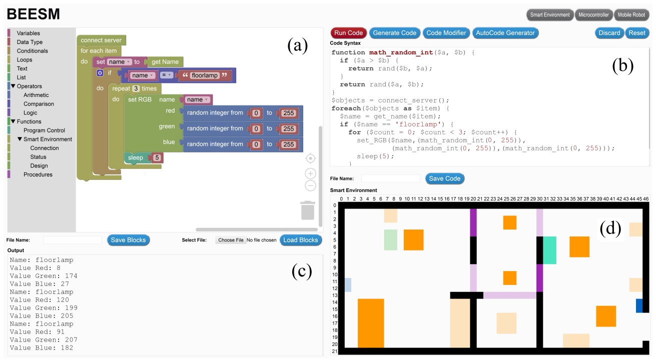
Primitives
In previous work, capabilities of smart devices were organized into primitives [6, 7, 11]. These primitives are defined as a set of custom blocks implemented in *BEESM* to allow users to work with them and see the reactions of smart objects in real-time. In this way, apart from predefined blocks, a set of custom blocks are provided for all primitives with inputs, outputs and types of connections. In a smart environment, the smart objects have a set of primitive behaviors, such as changeable status or being creatable by users. Robot's behavior and capabilities were implemented into primitives. Each primitive can be called through a function such as navigating the robot to a location. These primitives generate PHP code syntax and enable users to work with them in order to see the reaction of robots and smart devices. Furthermore, primitive Arduino's behavior is wrapped in a set of blocks which generate Arduino code. Users can then integrate these primitives into general purpose of programming languages like variables, conditionals, loops,

functions, etc.

Graphical interface
BEESM is divided into three parts which are *Smart Environment*, *Mobile Robot* and *Microcontroller*. Based on the interaction techniques which are addressed in [9, 10, 13], four panels were designed in all three parts. These panels enable users having a full vision of the Blocks, code syntax, output of the code, and a 2D view of the smart environment (see Figure 1). The four panels are:

(a) *Blockly UI* includes a simple block-based workspace and a toolbox that contains predefined and customized blocks. The workspace is used to reduce the syntax errors and provide the programming language constructs.
(b) A *Code panel* demonstrates the generated code by blocks to users and enables them to edit the code syntax. To this end, a simple version of programming languages used to help users identify mistakes.
(c) An *Output panel* shows program's output and errors for debugging purposes.
(d) *2D graphical view* of the smart environment and robot's position provides users a picture of how the objects status are changed based on their program.

Compilation and execution
A useful feature of *BEESM* is the ability to run the code either from Blockly workspace or from the *Code* panel. Users can change the code by clicking on the *Modify Code* button and run it from the *Code* panel instead of the Blockly workspace. However, the panel will be changed back to the code generated by blocks, if the *Generate Code* button is clicked again (see Figure 1). *BEESM* allows users to toggle between blocks and code syntax to familiarize with it. This enables users to program smart environments, microcontrollers, and mobile robots using both blocks or directly with source code.



```
function math_random_int($a, $b) {
  if ($a > $b) {
    return rand($b, $a);
  }
  return rand($a, $b);
}
$objects = connect_server();
foreach($objects as $item) {
  $name = get_name($item);
  if ($name == 'floorlamp') {
    for ($count = 0; $count < 3; $count++) {
      set_RGB($name,(math_random_int(0, 255)),
              (math_random_int(0, 255)),(math_random_int(0, 255)));
      sleep(5);
    }
```

```
Name: floorlamp
Value Red: 8
Value Green: 174
Value Blue: 27
Name: floorlamp
Value Red: 120
Value Green: 199
Value Blue: 205
Name: floorlamp
Value Red: 91
Value Green: 207
Value Blue: 182
```

Figure 1: *BEESM* Graphical UI.

BEESM Architecture
As shown in Figure 2, *BEESM* is divided into the three layers which are *Frontend*, *Middleware* and *Backend*:

- The *Frontend* includes *BEESM* user interface. Users can generate either PHP code or Arduino code using a set of blocks or directly using code syntax. In the *Smart Environment* and *Mobile Robot* parts, each block generates PHP code based on their inputs and outputs. Arduino code is generated in the *Microcontroller* part in order to upload the code directly towards microcontrollers.
- The *Middleware* consists of three files containing a set of wrapper functions for the primitive behavior of smart objects, mobile robots and microcontrollers. These wrapper functions help to execute corresponding functions which are understandable for smart environments using OpenHab2 (a unified open-source home automation tool), Robot Operating System (ROS) and microcontrollers using CherryPy (an open-source web-framework).
- The *Backend* consists of OpenHab2 and CherryPy which includes exchangeable functions for the re-

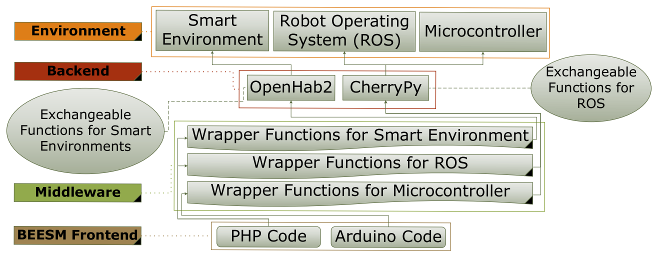

Environment
Backend
Middleware
BEESM Frontend
Smart Environment
Robot Operating System (ROS)
Microcontroller
OpenHab2
CherryPy
Exchangeable Functions for ROS
Exchangeable Functions for Smart Environments
Wrapper Functions for Smart Environment
Wrapper Functions for ROS
Wrapper Functions for Microcontroller
PHP Code
Arduino Code

Figure 2: *BEESM* Architecture.

spective environment. Currently, OpenHab2 contains the functions for smart environments, and CherryPy contains the functions for mobile robots based on ROS. For microcontrollers, CherryPy is used as a helper for Google Blockly to receive the generated code via an HTTP request and upload it on microcontrollers.

Overall *BEESM* is a customizable tool to assist inexperienced users and novice programmers to learn and build PHP code in order to program smart environments and mobile robots. Furthermore, Arduino code is used to program microcontrollers. The code for program is generally done by generating it for blocks which are at the top level of the program. When the user runs the program, the generated code is sent to our *Backend* through *Middleware*. In this approach, smart environments can be controlled through mobile robots and microcontrollers. They can communicate with each other either directly or via *Middleware*. In an educational tool, this helps educators to give different accessibility to users based on their needs and prior knowledge. *BEESM* backend consists of exchangeable functions for corresponding environments. Educators and engineers can change these functions to adapt and customize *BEESM* to other smart environments and mobile robots. In the *Microcontroller* part, the generated code is directly uploaded into the board using our backend as a helper for Google Blockly. *BEESM* architecture and its primitives provide a hassle-free environment for educators and engineers. They are able to use *BEESM* for educational purposes for different smart environments and devices. *BEESM* can help users via using graphical blocks to generate code syntax instead of memorizing it. This approach reduces syntax errors and eases the manipulation of code structures.

Future Work

This approach was tested on a smart ambient assisted living lab flat, a TurtleBot3 as well as on Arduino Uno and WEMOS D1 boards. A necessary feature that needs to be added to *BEESM* is to enable users to control mobile robots using different microcontrollers such as WEMOS D1 boards. One of the most important future contributions is the evaluation of ease of use and expressiveness of *BEESM* amid the users with and without prior programming knowledge. Furthermore, *BEESM* does not yet provide real-time hints for debugging purposes to help users, but developing such an implementation is future work.

Conclusion

We presented the design and development of *BEESM*, a *Block-based End-user programming application for SMart Environments*. *BEESM* allows inexperienced users and novice programmers to rapidly prototype and experiment with new applications for smart devices and environments.

In conclusion, *BEESM* is designed to support real-world programs and educational tasks. *BESSM* is designed to be a block-based rapid programming tool which provides a hassle-free environment for educators and engineers to make it compatible with different smart environments and devices for educational purposes.

Acknowledgement
This work was funded by the German Federal Ministry for Education and Research (BMBF), grant number 01FP1613.

REFERENCES

1. David Bau. 2015. Droplet, a blocks-based editor for text code. *Journal of Computing Sciences in Colleges* 30, 6 (2015), 138–144.
2. David Bau, Jeff Gray, Caitlin Kelleher, Josh Sheldon, and Franklyn Turbak. 2017. Learnable programming: blocks and beyond. *Commun. ACM* 60, 6 (2017), 72–80.
3. Brian Broll, Akos Lédeczi, Peter Volgyesi, Janos Sallai, Miklos Maroti, Alexia Carrillo, Stephanie L Weeden-Wright, Chris Vanags, Joshua D Swartz, and Melvin Lu. 2017. A visual programming environment for learning distributed programming. In *Proc. ACM SIGCSE Technical Symposium on Computer Science Education*. ACM, 81–86.
4. Anind K Dey, Raffay Hamid, Chris Beckmann, Ian Li, and Daniel Hsu. 2004. a CAPpella: programming by demonstration of context-aware applications. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 33–40.
5. David Harel, Assaf Marron, and Gera Weiss. 2012. Behavioral programming. *Commun. ACM* 55, 7 (2012), 90–100.
6. Justin Huang and Maya Cakmak. 2015. Supporting mental model accuracy in trigger-action programming. In *Proc. ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 215–225.
7. Justin Huang, Tessa Lau, and Maya Cakmak. 2016. Design and evaluation of a rapid programming system for service robots. In *The Eleventh ACM/IEEE International Conf. on HRI*. IEEE Press, 295–302.
8. Andrew J Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, and others. 2011. The state of the art in end-user software engineering. *ACM Computing Surveys (CSUR)* 43, 3 (2011), 21.
9. Andrew J Ko and Brad A Myers. 2006. Barista: An implementation framework for enabling new tools, interaction techniques and views in code editors. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 387–396.
10. Andrew J Ko, Brad A Myers, and Htet Htet Aung. 2004. Six learning barriers in end-user programming systems. In *Visual Languages and Human Centric Computing, 2004 IEEE Symposium on*. IEEE, 199–206.
11. Cecilia Martinez, Marcos J Gomez, and Luciana Benotti. 2015. A comparison of preschool and elementary school children learning computer science concepts through a multilanguage robot programming platform. In *Proc. ACM Conference on Innovation and Technology in CS Education*. ACM, 159–164.
12. Brad A Myers, Andrew J Ko, Sun Young Park, Jeffrey Stylos, Thomas D LaToza, and Jack Beaton. 2008. More natural end-user software engineering. In *Proceedings of the 4th international workshop on End-user software engineering*. ACM, 30–34.
13. Alexander Repenning and Andri Ioannidou. 2006. What makes end-user development tick? 13 design guidelines. In *End user development*. Springer, 51–85.