Blocks? Graphs? Why Not Both? Designing and Evaluating a Hybrid Programming Environment for End-users

Nico Ritschel
ritschel@cs.ubc.ca
University of British Columbia
Vancouver, British Columbia, Canada

Felipe Fronchetti
fronchettl@vcu.edu
Virginia Commonwealth University
Richmond, Virginia, USA

Reid Holmes
rtholmes@cs.ubc.ca
University of British Columbia
Vancouver, British Columbia, Canada

Ronald Garcia
rxg@cs.ubc.ca
University of British Columbia
Vancouver, British Columbia, Canada

David C. Shepherd
dshepherd@lsu.edu
Louisiana State University
Baton Rouge, Louisiana, USA

ABSTRACT

Many modern end-user development environments support one of two visual modalities: block-based programming or data-flow programming. In this work, we investigate the trade-offs between the two modalities in the context of robotics tasks. These often contain both aspects that are better solved with blocks and others that best fit data-flow programming. To address this style of task, we present and discuss two novel programming environment prototypes, one purely block-based and one a hybrid of blocks and data-flow programming. We compare the designs through a controlled experiment with 113 end-user participants, in which we asked them to solve programming and program comprehension tasks using one of the two environments. We find that participants preferred the hybrid environment in direct comparison, but performed better across all tasks and also reported higher usability ratings for blocks.

ACM Reference Format:
Nico Ritschel, Felipe Fronchetti, Reid Holmes, Ronald Garcia, and David C. Shepherd. 2024. Blocks? Graphs? Why Not Both? Designing and Evaluating a Hybrid Programming Environment for End-users. In *2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3639478.3643101

1 INTRODUCTION

Millions of people write code as part of their job, but the vast majority of them are *end-users*, who have received no formal programming-related education and only limited training [5]. Due to this lack of formal training, end-users are ill-equipped to use traditional programming languages or tools. Instead, they rely on domain-specific tools and languages that are designed to be easy to learn and use [1].

End-user tools typically use rich user interfaces that leverage visual aids and notations, making them expensive to develop from scratch. To save development cost and effort, many tool developers build on established frameworks that are suitable for a wide range of programming domains. *Block-based programming* [11, 3] and *data-flow programming* [2, 10] are two of the most commonly used frameworks for this purpose. The former originates in computer science education [6] and uses graphical puzzle blocks that can be assembled through drag-and-drop to edit program syntax. The latter has a long history in industrial languages [4, 7], and visualizes programs as directed graphs that represent information flow.

Previous work has explored the use of blocks and data-flow graphs in different domains. Data-flow representations tend to be used for data and event processing, because they are well-suited to illustrating the step-wise evaluation of expressions. Blocks on the other hand are typically used to used to describe imperative sequences of commands, like animations [6] or robot manufacturing steps [11]. However, no previous work has explicitly discussed the potential trade-offs between the two modalities.

Ideally, an end-user programming tool should support creating both imperative and expression-based code. When users define *what* a program should do, it is often easier to do so in an imperative style, whereas defining *when* it should do it often requires them to create logical, math-like expressions. In this work, we consider a concrete use case where both aspects of a program are important and can become complex enough to challenge end-users: programming a mobile robot that moves between workstations and operates machinery. Existing approaches have also demonstrated that block-based programming can support end-users as they define *what* actions such a robot should perform [11, 9]. However, operating machines requires more careful planning of *when* a robot should execute these operations. Existing end-user robotics tools provide only very limited support this type of planning.

2 APPROACH: TRIGGER BLOCKS VS. GRAPHS

One established way to support end-users in the programmatic planning of tasks are *triggers*, a simplified form of event-based programming [8]. Defining triggers of non-trivial complexity requires creating a nested logical expressions, which can be evaluated to determine when a trigger is executed. Unfortunately, traditional block-based programming tools do not support this style of programming well. Blocks limit how users can format nested expressions, forcing them into a single line. The block structure further makes it difficult to discern the structure of an expression.

ICSE-Companion '24, April 14–20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0502-1/24/04…$15.00
https://doi.org/10.1145/3639478.3643101

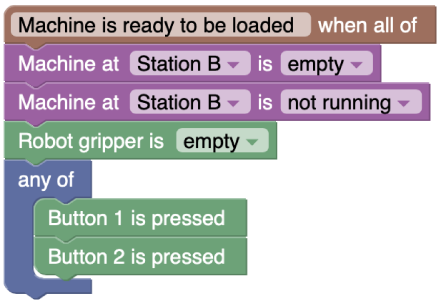

Figure 1: A block-based trigger expression for a mobile robot. The computational flow of the nested sub-expressions is not obvious for beginners.

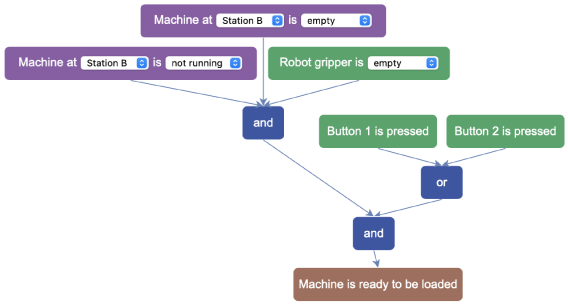

Figure 2: The same program as in Figure 1 represented as a data-flow graph. The edges make the data-flow explicit and programmers can manually group and arrange nodes.

In this work, we present a modified version of block-based programming that better supports nested expressions, as shown in Figure 1. This version spreads expressions over multiple lines, making them easier to read and edit. However, remaining within the structure of blocks limits how clearly the dependencies within an expression and the flow of its evaluation can be represented.

To overcome the remaining limitations of block-based expressions, we further consider the alternative that is using a true *hybrid* system: one that uses blocks for imperative code and data-flow graphs for trigger expressions. Figure 2 illustrates how trigger expressions are represented in this design alternative. Unlike blocks, this graph-based design allows users to group and arrange sub-expressions freely and trace their execution flow top-to-bottom. However, this design is only suitable for side-effect-free expressions and therefore still requires blocks to represent other, imperative parts of a program. It is also less space-efficient and potentially more difficult to edit as users have to manually adjust edges and connections when moving or replacing nodes in the graph.

3 EVALUATION

To investigate and compare the previously presented design options, we created prototype environments that implement each approach.

3.1 Experimental Design

To compare our two design prototypes, we conducted a controlled experiment. 113 end-users were recruited via the *Prolific* online platform to compare the two prototypes. Each participant was trained to use their respective environment and were asked to complete two complex tasks that required writing both nested expressions for triggers and imperative robot code. We then evaluated their ability to comprehend isolated, complex examples of triggers that were represented as blocks or as a graph. Finally, participants rated their environment individually and in comparison to the alternative.

3.2 Experimental Results

For the first of the two programming tasks, we find that 71% of the blocks-users and 51% of the hybrid-users successfully completed the task. For the second programming task, the overall performance was worse and only 54% of the blocks-users and 37% of the hybrid-users solved the task successfully. For both program comprehension tasks, we saw a similar difference success rates between the groups (85% vs. 65% for Q1, 78% vs. 62% for Q2). These results are complemented by the participants' own ratings, which were also substantially higher for those who used blocks. In contrast to our other findings, when we asked participants to directly compare the environments, both groups showed a preference for the hybrid environment. This observation might be caused by the appeal of data-flow graphs as a visually richer, supposedly more readable representation for code.

Our evaluation provides initial evidence that blocks might have a greater benefit for end-users than data-flow graphs. Though we performed our evaluation on the domain of robotics, we believe that our findings might transfer to related end-user domains, such as home and web automation, and game development. We believe that these observations can inform the design of future end-user tools in those areas, as well as additional research on how to create new, novice-friendly interface designs.

REFERENCES

[1] Brian James Dorn. 2010. *A case-based approach for supporting the informal computing education of end-user programmers.* Ph.D. Dissertation. Georgia Institute of Technology.
[2] Jody Condit Fagan. 2007. Mashing up multiple web feeds using yahoo! pipes. *Computers in Libraries*, 27, 10, 10–17.
[3] Mateus Carvalho Gonçalves, Otávio Neves Lara, Raphael Winckler de Bettio, and André Pimenta Freire. 2021. End-user development of smart home rules using block-based programming: a comparative usability evaluation with programmers and non-programmers. *Behaviour & Information Technology*, 1–23.
[4] John L Kelly, Carol Lochbaum, and Victor A Vyssotsky. 1961. A block diagram compiler. *The Bell System Technical Journal*, 40, 3, 669–678.
[5] Andrew J Ko et al. 2011. The state of the art in end-user software engineering. *ACM Computing Surveys (CSUR)*, 43, 3, 1–44.
[6] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The scratch programming language and environment. *Transactions on Computing Education (TOCE)*, 10, 4, 1–15.
[7] J Paul Morrison. 1994. Flow-based programming. In *Proc. 1st International Workshop on Software Engineering for Parallel and Distributed Systems*, 25–29.
[8] Steven Ovadia. 2014. Automate the internet with "if this then that" (IFTTT). *Behavioral & social sciences librarian*, 33, 4, 208–211.
[9] Nico Ritschel, Felipe Fronchetti, Reid Holmes, Ronald Garcia, and David C Shepherd. 2022. Can guided decomposition help end-users write larger block-based programs? a mobile robot experiment. *Proceedings of the ACM on Programming Languages*, 6, OOPSLA2, 233–258.
[10] Brenden Sewell. 2015. *Blueprints visual scripting for unreal engine.* Packt Publishing Ltd.
[11] David Weintrop, David C Shepherd, Patrick Francis, and Diana Franklin. 2017. Blockly goes to work: block-based programming for industrial robots. In *Blocks and Beyond Workshop (B&B)*, 29–36.