Ardestan: A Visual Programming Language for Arduino

Hiroki Nishino
Dept. of Industrial Design
Chang Gung University
Taoyuan, Taiwan
hiroki.nishino@acm.org

ABSTRACT

This paper describes Ardestan, a visual programming language (VPL) for Arduino currently under development. The language is designed for art and design students as novice programmers in mind. While multitasking and event scheduling are essential for interactive prototypes, novices often face a difficulty in implementing these features. By borrowing language design from Pure Data, a VPL for interactive music, Ardestan facilitates the implementation of multitasking and event scheduling, while it also generates C++ code for a standalone Arduino system. Such features would be beneficial to support prototyping activity by art and design students in undergraduate interaction design courses.

CCS CONCEPTS

• Software and its engineering → Visual Languages

KEYWORDS

Visual programming; novice programming; physical computing; Arduino; rapid prototyping; STEM education; interaction design.

ACM Reference format:

Nishino, Hiroki 2018. Ardestan: Visual programming language for Ardino. In *Proceedings of ACM SIGGRAPH '18 Posters. ACM, New York, NY, USA, 2 pages.* https://doi.org/10.1145/1234567890

1 INTRODUCTION

While existing novice programing environments for Arduino can ease prototyping activity for simple tasks, novices often face a significant difficulty when multitasking and event scheduling must be involved. As these are important features to develop interactive systems, the lack of them can significantly damage the quality of prototypes. Fortunately, as both of these features are essential also for music application, there exist visual programming languages (VPLs) for computer music that can significantly facilitate the implantation of multitasking and event scheduling. With extension modules, such a VPL can control an Arduino device where the associated firmware is installed.

However, an Arduino prototype built on such a VPL cannot work standalone and it must always involve a personal computer to control an Arduino device. This is clearly not quite favorable, not just when many Arduino devices must be utilized, but also when a standalone system is preferable for certain artistic/design concepts (for example, fashion technology or IoT products, etc.)

With consideration of such needs in prototyping activity by art and design students, we are currently developing Ardestan, a visual programming language for Arduino. While its language design mostly relies on the node-based programming concept as seen in Max and Pure Data, which facilitates multitasking and event scheduling, it also generates C++ source code for a standalone Arduino system. Such a language would be beneficial for art and design students without programming expertise in undergraduate interaction design courses.

2 RELATED WORK

There already exists programming environments that target novice programmers for Arduino. For example, BlocklyDuino [1] is a block-based coding environments that translates a visual program to an equivalent C++ code. Pduino [6] is an extension modules for Pure Data [4], node-based VPLs for interactive music to control an Arduino device. Yet, while such a programming environment facilitates multitasking and event scheduling, a standalone Arduino system can't be implemented on Pduino. Visuino [7] and XOD [8] are similar node-based VPLs, yet they can generate C++ source code for Arduino. However, the features for abstraction (subpatching) and event scheduling seems still weak compared to interactive music VPLs.

3 DESCRIPTION OF OUR WORK

Considering such practical needs to support prototyping activity by art and design students in undergraduate interaction design courses, we developed Ardestan, a visual programming language for Arduino. Its language design is largely borrowed from Pure Data so that it can facilitate the implementation of multitasking and event scheduling, yet its visual programs are translated into C++ source code so that the software can allow an Arduino system to run standalone. The current version of Ardestan is written in Java to support multiple platforms (Mac/Win/Linux).

Figure 1 (left) shows an example to blink one LED with the period of 700msec on followed by 300m sec off. When an Ardestan program is loaded, similarly as in Pure Data, loadbang objects emit a *bang* message. In Figure 1 example, the bang message is received by the connected 'symbol' object, which emits a *start* symbol from its outlet. The symbol is then received by the 'metro' object, which starts repeatedly emitting a *bang* message with the given period (1000 msec). The *bang* message is received by an 'int' object (with the argument: 1) and a 'delay'

object. The 'int' object sends out an integer value: *1* to the 'dout' object (digital out), which set the pin #2 high. The 'delay' object passes through the *bang* message with a 300 msec delay, which is received by another 'int' object, resulting an integer value: *0* to be sent out. When the 'dout' object receives it, it turns off the pin #2 low. This process is repeated every 1000 msec cycle. When compiled, this visual program is translated into C++ source files for Arduino.

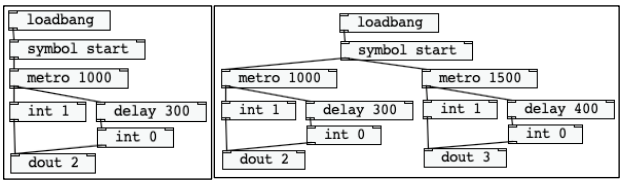

Figure 1: Code examples in Ardestan to blink one LED (left) and blink two LEDs with different periods (right)

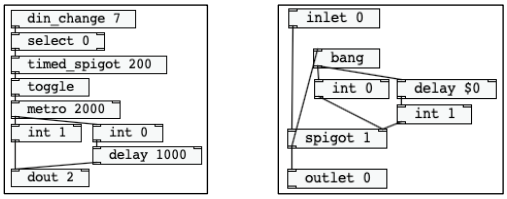

Figure 2: A subpatching example in Ardestan. The left patch is using the *timed_spigot* subpatch on the right.

Figure 1 (right) illustrates an example in Ardestan to blink two LEDS at the different periods (700 msec on/300 msec off and 1100 msec on/400 msec off). As shown, the implementation can be almost done just by copying the Figure 1 (left) code. Ardestan also supports 'subpatching' for abstaction as provided in Pure Data. Figure 2 example shows an example of subpatching. The Figure 3 right example, which is saved with the filename: *timed_spigot.ard*. is utilized as an object in the left patch. As shown, by using 'outlet' and 'inlet' objects, a subpatch can receive and output messages. Subpatches can receive arguments by using symbols such as $0, $1, $2…, the number of which stands for the index of the arguments.

4 DISCUSSION

The introduction of multitasking and event scheduling may create a steep learning curve for novices. For instance, Figure 3 illustrates a BlocklyDuino example equilvalent to in Figure 1 (right). To blink two LEDs with different periods, the code involves state variables for LEDs (*led1_on* and *led2_on*) and timestamps for event scheduling (*timestamp1* and *timestamp2*). Such an introduction of new concepts and programming patterns may lead to a difficulty in both comprehension and implementation for novices. On the contrary, as its language design hides such complexity, the Figure 1 (right) Ardestan example can be simply implemented by reusing the knowledge they acquired to blink one LED. As shown in Figure 1 (right), there is no introduction of new concepts and programming patterns in Ardestan, even when multitasking is involved and the number of objects in the program are significantly smaller.

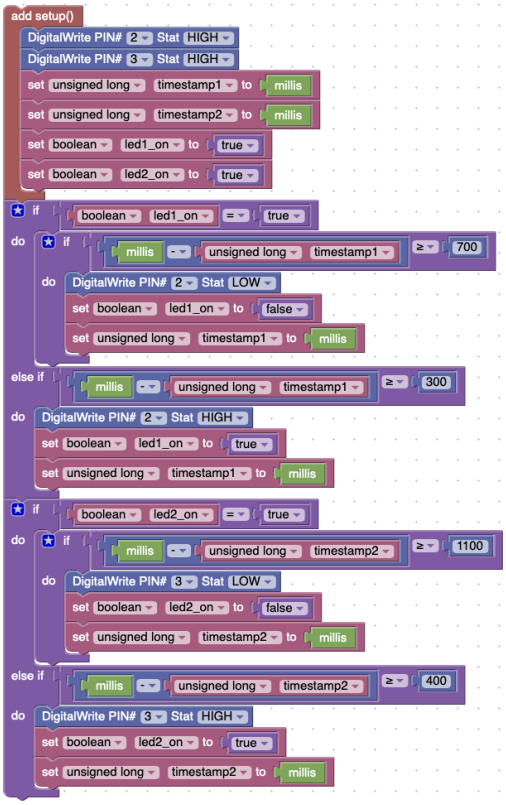

Figure 3: A BlocklyDuino example to blink two LEDs with different periods (equivalent to the Figure 2 (right) example).

5 CONCLUSION AND FUTURE WORK

We described Ardestan, a VPL for Arduino. The language is designed to facilitate the implementation of multitasking and event scheduling, while it also generates code for standalone Arduino systems. Such features can be helpful to prototyping activity by art/design students in undergraduate interaction design courses. Ardestan provides basic objects such as digital I/O, analog I/O, serial output, arithmetic and relational operators, etc. These are already enough to support basic tutorials for physical computing with Arduino. Yet, more objects are planned to be developed so that it can further facilitate prototyping activity.

ACKNOWLEDGMENTS

The author acknowledges the support of the Ministry of Science & Technology, Taiwan under Grant 107-2218-E-182-005-MY2.

REFERENCES

[1] Fred Lin. 2015. *GitHub – BlocklyDuino/BlocklyDuino*. Retrieved Mar 26 from https://github.com/BlocklyDuino/BlocklyDuino.
[2] Marius Schebella. 2007. Pduino and other AArduino interface for Pd. In *Proc. of Pd. Convention*.
[3] Miller Puckette. 1997. Pure Data. *Proc. Of Int'l Computer Music Conference*.
[4] Mitov Software. 2017. Visuino – Visual Development for Arduino. Retrieved Mar 26 2019 from https://www.visuino.com
[5] XOD Inc. 2017. *XOD*. Retrieved Mar 26 2019 from https://xod.io