11-1-2016

A Browser-based IDE for the MUzECS Platform

Omokolade Hunpatin
Marquette University

Casey O'Hare
Marquette University

Ryan Thomas
Marquette University

Dennis Brylow
Marquette University, dennis.brylow@marquette.edu

A Browser-based IDE for the MUzECS Platform

Omokolade Hunpatin Casey O'Hare Ryan Thomas Dennis Brylow
Marquette University
MSCS Department – Cudahy Hall
1313 W. Wisconsin Ave.
Milwaukee, WI 53233
firstname.lastname@marquette.edu

***Abstract*—We report on a scalable, portable, and secure visual development environment for programming embedded Arduino platforms with Chromebooks in a successful secondary school computer science curriculum. Our web-based environment is part of the larger MUzECS project, an inexpensive replacement module for the Exploring Computer Science (ECS) course being widely deployed in United States high schools. Students use MUzECS to gain a deeper understanding of computing, through a set of blocks which provide appropriate abstractions for working with low-level hardware.**

MUzECS improves upon the existing curriculum module by reducing the hardware cost by an order of magnitude, while still preserving the key ECS pillars of computer science content, student inquiry and classroom equity. Programming with visual blocks provides a more attractive tool for introductory courses than traditional approaches, and yet enables high-impact exploration activities such as building a series of embedded musical instruments.

The current work combines and modifies several existing tools to eliminate technical barriers on low-cost platforms like Chromebooks, such as the reliance on special block-based toolchains, remote compilation servers, or multi-stage transfers for student code.

1. Introduction

According to the US Bureau of Labor Statistics, nearly 500,000 new jobs will be created in computing over the course of the next 10 years [14]. Computer Science drives innovation throughout much of the world economy, but it remains marginalized throughout primary and secondary school in many countries. Exploring Computer Science (ECS) [5] is a secondary school course which is currently being adopted in many parts of the United States. ECS was designed from the outset to address persistent gaps in representation by women and minority ethnic groups observed in the computing field [10]. It is targeted to early high school students (ages 14-16), and is designed to work well in traditionally under-resourced schools. However, the Achilles' Heel of ECS has proven to be its sixth and final curriculum module, which has, until the latest revision, relied on engaging, but costly and proprietary, robotics kits.

The MUzECS Project [2] was launched to provide a low-cost alternative to the ECS robotics module, and has been field-tested by hundreds of students in a dozen different school classrooms since its deployment. The physical platform which MUzECS operates on is a combination of an Arduino *Leonardo* or an Arduino *Uno* - inexpensive, commercially available credit-card sized embedded computers - and a "shield" - a circuit board extension that plugs into the top of the Arduino board to provide additional peripheral hardware. The Arduino is commercially available, but we produce the MUzECS shields in-house and provide them at cost. In contrast with current offerings for ECS module 6, our platform is open-source and can be easily extended to work with a variety of specializations within secondary school computer science, all at a very low price.

Recent trends in educational technology have led to a growing number of schools investing in Chromebooks [8], thin client laptops that run a Linux variant and the Chrome internet browser. For schools, Chromebooks represent inexpensive machines with lower maintenance costs and few of the device driver, application compatibility and software virus problems inherent in other types of personal computer. For computer science educators, Chromebooks encourage reliance on cloud-based services, but present new technical barriers to installing traditional software development tools, such as integrated development environments (IDEs), compilers, and debuggers.

The first release of the MUzECS programming dialect leveraged the prior Ardublock system, which could operate on any platform capable of running Java applications. Chromebooks, due to their very nature, do not allow such applications to run.

In this work, we present a powerful, web-based graphical programming environment for Arduinos and MUzECS shields, capable of running on stock Chromebooks as well as virtually any platform compatible with the Chrome web browser. Our solution consists of a browser-based IDE for Google Chrome, and a Chrome extension which allows for client-side execution of users programs. This is portable to more platforms than prior work, scales to a larger number of students with reduced load on webservers, and closes several usability and security issues with prior work.

DOI reference number: 10.18293/DMS2016-043

1.1. Initial Solution

In designing a block-based IDE for wide deployment in high schools, it is necessary to ensure that many students can use the tool simultaneously (scalability), and that they do not access other students' programs (uniqueness/security.) Finally, we must ensure that our system can be used on the wide variety of platforms used in high schools (portability).

MUzECS's first software platform was based on Ardublock, a Java-implemented graphical development environment which translates blocks directly to Arduino code. Ardublock is a stable add-on to the widely used Arduino IDE, and it handles the uniqueness and scalability problems effectively, as all code is executed on the user's own machine. Furthermore, since Java is a platform-independent programming language, this solution was quite versatile, capable of running on traditional Windows, Mac, and Linux OSes. Schools in the Milwaukee area, however, have been gradually adopting Chromebooks, a low-cost and lightweight laptop. Unfortunately, Chromebooks are only able to run the Chrome browser and Chrome Applications, and cannot install general purpose software like the Ardublock toolchain.

1.2. Arduino

The Arduino is a small microcontroller board with a universal serial bus (USB) plug to connect to your computer and a number of connection sockets that can be wired to external electionics such as motors, relays, light sensors, laser diodes, loud speakers, microphones, and more. They can either be powered through the USB connection from the computer, from a 9V battery, or from a power supply [12].

Our MUzECS shield attaches to the GPIO pins on the Arduino Uno or Leonardo. Our MUzECS shield consists of four LEDS, four buttons, a piezo speaker, and a distance ping sensor (Figure 1). We made blocks to manipulate all of the peripherals on our MUzECS shield.

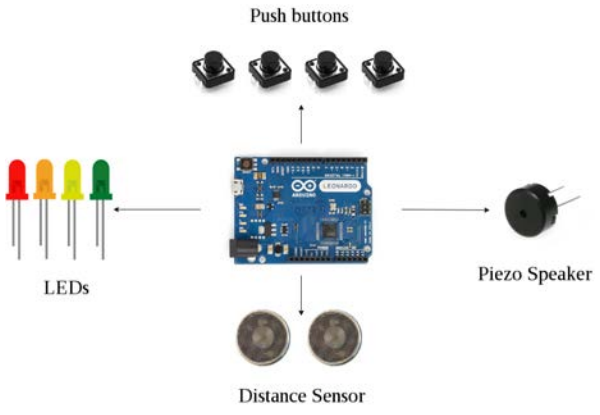

Figure 1. Peripherals for the MUzECS shield board

For a detailed breakdown of the capabilities of our different hardware platforms, see section 4.3: Hardware Support.

2. Related Work

Several platforms share the overall goals of the MUzECS project. Earlier versions of the ECS curriculum used LEGO Mindstorms robotics kits, which while highly versatile, were prohibitively expensive for many schools implementing ECS. Version 7 of the ECS curriculum will use more cost effective Edison Robots[3]. Like MUzECS, the Edison Robotics platform has a visual language designer, EdWare, where users assemble and connect blocks in a graphical environment to program. The Edison Robots themselves are LEGO robotics kits which are designed to move around. Like MUzECS' software, EdWare is free, and the Edison Robotics hardware is substantially cheaper than the LEGO mindstorms, ranging from $33-$50 per kit[3].

Code.org's CS Discoveries course is expected to make use of the Adafruit Circuit Playground[1], an all-in-one Arduino platform with similar peripherals to the MUzECS hardware, plus several attractive improvements, for an astonishing price below $20 per kit. At this writing, the Circuit Playground hardware is not yet widely available, (and the proprietary design cannot be replicated by third parties,) the full accompanying curriculum has not yet been released, the block-based tools for programming are progressing through beta versions, and pilot teachers have not yet received training for deploying the curriculum next spring.

Both the Edison and Circuit Playground alternatives were made available late enough in 2016 to miss the teacher training and deployment windows in the spring. The earliest opportunity for head-to-head comparison of these curricula in real ECS classrooms will thus be in spring 2017. In contrast, our work has been publically available for ECS teacher use since spring of 2015, and now will also be available on Chromebooks for the coming school year.

With MUzECS, Edison, and Circuit Playground all available for less than $60 per kit, the primary criteria for teachers to differentiate upon is the power and flexibility of the platforms. To that end, while MUzECS is the most expensive of the three, it is also the most open-ended with the clearest path forward for subsequent high school courses that would teach more advanced concepts, transition to text-based programming langauges, or be extendible with new shield hardware peripherals.

OzoBlockly [4] is another web IDE based upon Blockly. The OzoBlockly IDE pairs with the proprietary "Ozobot Bit," a small robot, which can move atop a surface using autonomous LEDs, sounds and infrared proximity sensing. OzoBlockly has a unique and creative way to load a program to the OzoBot Bit. Instead of using the serial port on a computer, the OzoBot Bit can simply be placed on the screen of the computer and identify a sequence of flashing lights as a program. This allows for a very simple upload process to a device for the user and expands functionality to mobile devices that otherwise could not program external hardware. OzoBlockly focuses on small games to entice

younger users, while MUzECS approaches the user from a musical perspective. Additionally, the Ozobot Bit does not allow for direct human interaction, unlike our MUzECS shield.

3. MUzECS Blocks

Our MUzECS block dialect was made to be translated to code that would run on our Arduino with peripherals. Designing the blocks has proven to be a key challenge in crafting MUzECS. We wanted our blocks to be easy to use, visually appealing, and we wanted each to manipulate a single peripheral on the MUzECS shield. The MUzECS blocks were made not only with the goal of teaching core computer science concepts, but also with the goal of facilitating the natural transition from block to text-based programming.

3.1. Our blocks

We based our visual programming dialect on Google's open-source, visual-block based programming environment called Blockly [9]. Our block-based interface works as follows: users attach blocks together in an online interface, in the Google Chrome browser. All blocks which the users assemble must be placed inside a main control block which reads "program" and has two slots: "setup" and "loop". The "setup" slot is a place for the user to initialize different hardware components on their board. The "program" slot is where the user assembles the blocks for logic and instructions.

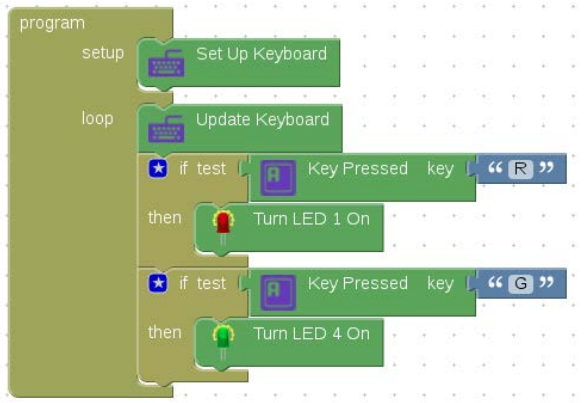

Figure 2. Example program which uses both the setup and loop slots of the program block

When approaching this problem, we had the option of doing all setup in the background, silently, or making it explicit. We made the decision to make setups explicit to help facilitate the transition to text-based languages, where initialization is a key concept. We should note that, when taking this approach, we chose to include simple error messages if the initialization is not performed. This stands in contrast to a programming language like Scratch, which does not include error messages. We go even further with the idea of initialization with the keyboard blocks, where "Set up Keyboard", "Update Keyboard", and "Key pressed" are all individual blocks. We chose to separate these actions each into their own functions rather than shadowing the behavior of the computer. We believe that these three separate blocks better help students understand the nuances of initialization and different objects. Another design decision that we made was to create drop-down menus for most of our operators, so that after a block has already been placed in the interface, it can be easily changed to a different operator without reconstructing the entire section.

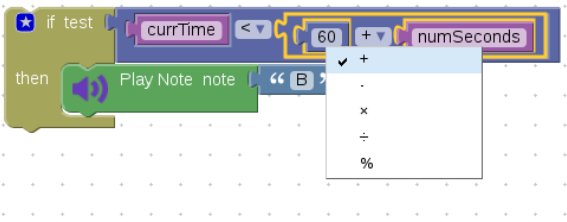

Figure 3. Clicking on the drop-down menu on the arithmetic operator lets the user select another operator

We have also designed our block-based interface to run in sequence, from top to bottom, and not to make our language event-driven. For more on this, see section 3.3 - Transition to Text-Based Languages.

We re-implemented the MUzECS dialect of Ardublock and some original Ardublock blocks to make the platform browser-based. Blockly's JavaScript API was used to create blocks, which already provided mechanisms for snapping blocks into place and generating code from blocks. Each drawer of blocks has a distinctive organizing color; the blocks that we implemented fall into several drawers: Control, ECS, Variables/Constants, Math operators, Logic operators, Communications, Advanced Pins and Advanced Code.

The "Control" drawer contains blocks which represent control structures commonly found in text-based programming languages: loops, if-then statements, and functions. Our goal with each of these blocks was to introduce students to control structures that they may use if they continue with programming. The "ECS" drawer contains the blocks for manipulating components of our Arduino hardware. We have blocks for: turning LEDs on and off, playing musical notes, playing notes for specific durations, reading the distance sensor, detecting button presses, reading host computer keyboard input, and reading the accelerometer values.

Blocks from the Control drawer and the ECS drawer are used in virtually every program that a student creates, so we paid extra attention to making these blocks carefully, and abstracting appropriately. The variables/constants drawer contain blocks that are used to create constants and variables to be used throughout the program. The math and logic operators drawers contain blocks to perform mathematical operations, get random numbers, and compare quantities. As

students progress and become more advanced programmers, they often use these blocks more frequently. Finally, the communications and advanced drawers contain blocks that can be used for debugging and more advanced operations - few students use these blocks, but we included them in case the instructors or especially advanced students would like to explore the interface on a lower level.

In the world of programming, there are always multiple ways to approach a problem. We wanted to assure that students using our block-based interface have multiple approaches available to them, so we built a diverse set of blocks into our interface. For instance, we have a "Play note" and a "Play note for a given time" block, and a "Play frequency" and "Play frequency for a given time" block - four different approaches to the same task. This kind of block diversity can be found throughout our interface. We aim for our interface to give students a minimal number of blocks, with a maximal amount of expressive power. We seek to strike a balance where students can express complex actions with simple blocks, and they don't have to labor over a complex block-based interface to perform relatively complex tasks. A prime example of this principle in action is the "Get Distance" block, which abstracts a substantial amount of JavaScript code into a single block.

3.2. Visualization of Functionality

Figure 4. MUzECS blocks with pictures that are associated with the function of the block.

Our top design criteria for the language is that it be simple to use for introductory high school students. Many blocks in MUzECS have pictures associated with them that are related to functionality. For example, "turn LED 1 on" block has a bright red LED picture on it, and "turn LED 4 on" block has a bright green LED picture on it. Likewise, "Button Pressed button #" has a picture of a hand pressing a button and "No Tone" block has a picture of a speaker with no sound waves coming out of it.

Blocks are generally colored according to functional category, such as green for I/O operations, and black for the Advanced Code block that allows users to inject raw text-based code into their program.

3.3. Transition To Text-Based Languages

Designing a system such that it facilitates a transfer of knowledge is an important, but not an easy task. Many MUzECS blocks were created not only because they are necessary to make our visual language complete, but also because they are similar to necessary control structures found in popular text-based languages (TBLs). For example, we have opted to make our programming language primarily structured around a single thread of control, despite the fact that JavaScript, our underlying language, lends itself more to an event-driven paradigm. This deliberate design decision was made because the dominant Arduino toolchains do not include support for multi-threading runtimes.

This decision was a crucial part of our design. It is noteworthy to contrast our model with another popular block-based language, Scratch, which aims to be an easier alternative in teaching people how to program. Scratch is developed at the Massachusetts Institute of Technology and is primarily event-driven. Scratch has been noted to fall short in facilitating the knowledge transfer to text-based languages, and its design as an event-driven language has been cited as a key reason why this is the case [2][7]. Another concept in TBLs which Scratch has struggled to establish is that of initialization [7]. We believe that using a single thread of control will help students to understand initialization, but we have also made an additional effort to teach initialization by creating setup blocks for certain hardware components, as mentioned earlier.

4. Chrome-based Arduino

4.1. Curriculum

Our curriculum is built on the existing secondary school Exploring Computer Science (ECS) curriculum. The ECS curriculum has proven to be a successful way to teach computer science to underrepresented groups in the past [11], and we specifically designed our curriculum in line with the goals and methods of ECS. The ECS curriculum abides by three guiding values: equity, inquiry, and CS content. ECS strives to be equitable by verifying that students from all backgrounds have a fair shot at learning about computer science. In practice, this means ECS teachers must choose assignments which all students have an (approximately) equal chance at understanding - and not choosing assignments that require students to understand the rules of chess, or giving extra credit on a test for a question about a sci-fi movie, for example. ECS also holds inquiry as a guiding value, meaning that, in classrooms, students and teachers should always be focused on discovering and asking questions, instead of constantly making assertions. Finally, ECS holds Computer Science content as an important tenant of the course. This may seem obvious, since the course is, after all, a computer science course. But its emphasis as only one of three guiding values is perhaps the most telling aspect of how ECS is meant to be taught. ECS is meant to be as much equitable and full of inquiry as it is about computer

science. This plays into the design of our system as well. We strive for our system not to just to teach computer science to students, but to be equally accessible for all students, and for it to instill a yearning for more computer science education.

We are confident that MUzECS fits the bill that ECS provides, which is exciting, given the past success of ECS. Simply put, we know our curriculum works, because we build it on the shoulders of a proven course.

4.2. A Scalable Platform

Our foremost concern when approaching this problem was developing a software system which was scalable. The central issue that we needed to solve was that of compiling the Arduino code; previously, we hosted the server which compiled the programs and sent the compiled code back to the users. We were then faced with a choice: on one hand, we could stick with the same model and throw more resources behind the compilation server. If we created a decent distributed system, we might have been able to outsource our compilation servers to a cloud-services-provider such as Microsoft or Amazon. In retrospect, this probably would have been a viable option. We reasoned, however, that we would be doing better if we could completely reinvent the model; it would be best if there was no server-side compilation at all. We know that networks and communication are inherently prone to eventual failure, so we think it best to remove them from the process, or minimize their role, if possible.

Our research revealed that it is, in fact, possible to program an Arduino without compiling Arduino code. If the Arduino is flashed with Standard Firmata [6] firmware, any client-side programming language which has an implementation of Firmata can be used to program an Arduino. Programming languages like Python, Perl, Ruby, JavaScript, Java, and more all have libraries for Firmata. We chose to use JavaScript, and more specifically, the Johnny-Five robotics framework, to program our Arduino.

Johnny-Five works by executing the JavaScript code directly on the host machine - the code doesn't compile down to Arduino code. The JavaScript code is executed using Node.js - a JS runtime which is specifically designed to build scalable network applications. As the program is being executed on the host machine, basic I/O instructions are transmitted to the Arduino board. With the Arduino Leonardo, the board is required to be connected to the computer while the program is running, and the instructions are sent to the Arduino via USB Serial. One should note that when we take this approach, the Arduino must be connected to a computer to run programs - we are unable to upload programs and run them from a battery. We have found, however, that, in the classroom, this point is more or less negligible - students almost always have their Arduino plugged in to their computer anyways. Finally, requiring the Arduino to be physically tethered does not hinder student development because the MUzECS shield does not have any moving parts.

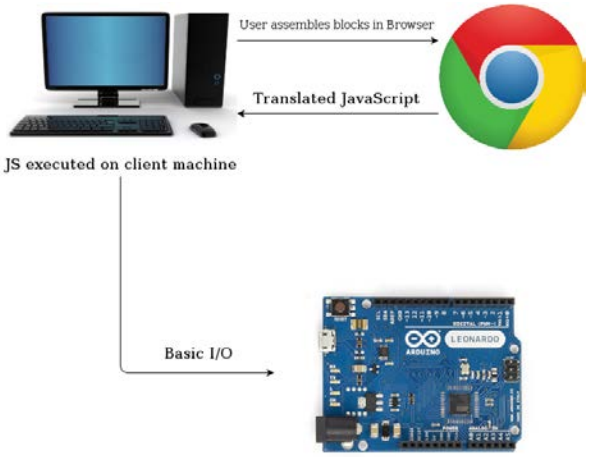

Figure 5. Design of our new system with Johnny-Five

Combining the different parts of this system was a relatively simple process. We used our existing block-based interface (Figure 4), which runs in the Chrome Application, and wrote the block definitions such that they translated directly to JavaScript code which satisfies the Johnny-Five framework. Our new architecture also works completely offline, once the application is downloaded.

Altogether, we believe our current software platform satisfies our original three considerations. It ensures uniqueness and security - we know that a student is only capable of running programs that they wrote on their own computer, on their own Arduino, because it must be physically connected.

Our platform is considerably more scalable than it ever has been. The burden of compiling programs has been shifted from the server to each user's computer. If a substantial influx of schools begin using our platform, we believe that our newly designed system will be able to manage the load with great efficiency. Finally, we believe our software platform is highly portable - perhaps as portable as is even possible for a modern software system. It is capable of running on any computing system that can download Google Chrome - meaning that our IDE can run on Chromebooks, Windows systems, Macs, and even the majority of Unix systems - again, virtually every computing system that is in use by modern high schools. By creating this software system, we believe we have combined the positive aspects of the previous versions of MUzECS and introduced new ones to make a secure, scalable, and portable platform.

4.3. Hardware Support

When the MUzECS platform was first launched, the only hardware that we supported was our own MUzECS shield (Figure 6) for the Arduino Leonardo.

Our shield was designed with a few considerations in mind; the foremost was to be cost-efficient. This is because we sought to design a cheap alternative to the most common

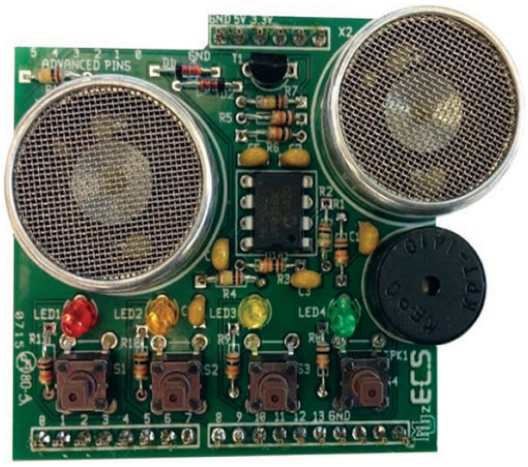

Figure 6. MUzECS shield

sixth module of ECS, which is incredibly expensive. Our design, uses a small set of peripherals - a speaker, four LEDs, four buttons, and a distance sensor. In order to avoid driving our cost up, we elected to build a third-party distance sensor into our design, which, consequently, does not have wide hardware or software support.

All of the hardware on our original MUzECS shield is still functional, even in our new model. We did, however, have to extend the firmata protocol in order to write JavaScript code for our distance sensor (refer to section 3.4 - Extension to Firmata Protocol for more). Still, the scope of what our shield hardware can do is somewhat limited. It should be noted that while the MUzECS package isn't the cheapest on the market, it has a wider range of hardware compatibility than other, similarly priced, options. The Edison Robot - a new installment in ECS curriculum v7 - is ten dollars cheaper than MUzECS, but it lacks the versatility of the Arduino Leonardo. Additionally, since the Edison Robot is a new addition to ECS, we have been unable to observe its effectiveness within the curriculum. We will be unable to present any sort of comparison between MUzECS' effectiveness and the Edison's effectiveness until Spring 2017, when teacher's use the platforms side-by-side in the classroom.

The Adafruit Circuit Playground (CP) is another option which has unique peripherals and is cheaper than our MUzECS package. Unlike the Arduino Leonardo, the CP (which is based on the Arduino *Flora*) has all of the necessary peripherals built-in. On the Circuit Playground, a circular board less than 2 inches in diameter, there are 10 RGB LEDs (all capable of emitting any RGB color), a piezo speaker, a triple-axis accelerometer, a light sensor, a sound sensor, a thermometer, and 8 capacitive touch pads which also act as general purpose input/output (GPIO) pins. The CP provides another low-cost alternative to more expensive robotics kits, and we plan to fully support the board with a set of blocks and curriculum in the near future.

4.4. Extension to Firmata Protocol

The open-source Firmata project provides a flexible protocol for remotely controlling a variety of embedded platforms from a tethered personal computer [13]. When an Arduino is imaged with the Standard Firmata sketch, a variety of programming language libraries can be used to issue platform-dependent instructions to the embedded hardware via a serial connections such as USB. The existing Firmata infrastructure was already suitable to manage most of the peripherals of the MUzECS hardware.

In this work, we have extended the Firmata protocol to support the MUzECS hardware ultrasonic distance sensors, a low-cost analog circuit not generally found on other platforms. Leveraging Firmata in our design has a number of advantages. The protocol is lightweight, requiring only 3 data payload bytes to set one or all the digital pins on the board [13]. Additionally, the Standard Firmata is the default firmware installed on every Arduino board, so it is readily available to students for upload to the Arduino. Our extended version of the Standard Firmata firmware, which we call MUzECS Standard Firmata, is similarly open-sourced, and included upload instructions for instructors and students.

Use of the Firmata protocol allows us to remove server-side compilation of the Arduino code, using the client local machine to run MUzECS block programs on the Arduino. The Standard Firmata is C++ Arduino code that runs on the Arduino and follows the client-server model, in which the Firmata sketch is the server running on the Arduino, and a client running on the user's computer issues commands to the embedded board.

Under the hood, the client sends *SysEx* messages to the Firmata server, which then executes actions on the Arduino. The standard command *SysEx* messages begin with a start byte (0xF0) and end with an end byte (0xF7). In between the start and end byte are 7-bit bytes which contain the commands one wants to send to the Firmata server. (See revelant segment of Firmata Protocol grammar in Figure 7.)

The Johnny-Five middleware doesn't directly support the MUzECS hardware ultrasonic distance sensor, but can pass extension commands through the existing Standard Firmata interface.

Our extension to the Firmata code base and protocol adds direct support for the MUzECS ultrasonic range finder. A new command type, *GetDistance*, was added to the existing *SysEx* command with byte 0x02. The client sends a *GetDistance SysEx* command to the MUzECS Standard Firmata firmware. After the firmware recieves the *GetDistance* command, it runs the code that activates the distance sensor. When the distance code completes, it sends the *GetDistance SysEx* command back to the client with the distance integer split in three 7-bit bytes. Among other low-level details, the Firmata code for handling the ultrasonic distance sensor performs smoothing of the data using a moving average, an important noise-cancelling step that allows block-based programs to produce useable musical input data.

<*SysExMessage*> → <*StartSysEx*> <*SysExCommand*> <*Data*>* <*EndSysEx*>
<*StartSysEx*> → 0xF0
<*SysExCommand*> → 0x00 - 0x7F
<*Data*> → 0x00 - 0x7F
<*EndSysEx*> → 0xF7

Figure 7. A BNF grammar of the format of *SysEx* messages

5. Future Work

One of our biggest concerns that we sought to address throughout this entire project was making MUzECS modular and reusable. First, we created our software to be modular by being able to support multiple hardware platforms. With new boards come new opportunities to modify our curriculum and deliver it in new and exciting ways. Furthermore, our open-source collaborators on this project have developed ways to communicate with Arduino via bluetooth, wi-fi, and even raw TCP sockets. These are all exciting possible extensions that we could make to MUzECS in the future.

Beginning in the spring of 2016, we deployed MUzECS into several pilot high schools in the Milwaukee area. We have been collecting survey data regarding students' usage of blocks. In the future, we will need to analyze this survey data and make appropriate improvements to our platform, on both the hardware and software side. It may also be a good idea to build into the system a mechanism for collecting block usage data automatically as we scale it out even further.

6. Conclusion

MUzECS's original goal was to create a cost-effective, block-based platform for the sixth module of the Exploring Computer Science curriculum. This paper presents technical solutions that extend this work to an increasingly common device in school classrooms, the Chromebook. The resulting system retains the carefully tuned block-based programming environment that has been specially adapted to hardware and curriculum widely used to broaden participation in introductory computer science courses. The contributions of this new version include improved scalability and security over previous browser-based solutions, with the advantage of greater flexibility for expansion to more advanced coursework than several alternative systems that will be piloted in ECS classrooms in spring of 2017.

Prior work studying the effectiveness of student learners transitioning from block-based languages to text-based languages has identified shortcomings related to understanding of initialization. Our current system includes decisions in the block language deliberately designed to address these problems, smoothing the transition to text-based.

The next steps in this work are to complete data collection from actual ECS classrooms that are using MUzECS on Chromebook, and to compare usability and student learning with alternative platforms.

Acknowledgements

The MUzECS project is supported in part by the National Science Foundation, grants CNS-1339392 and ACI 1461264. Our thanks to Rick Waldron and the Johnny-Five contributors, and Luis Montes for his contributions to the Chrome implementation. The anonymous VLC reviewers provided extensive feedback that improved the final version of this paper.

References

[1] Adafruit. Circuit playground, 2016. https://www.adafruit.com/product/3000.

[2] M. Bajzek, H. Bort, O. Hunpatin, L. Mivshek, T. Much, C. O'Hare, and D. Brylow. Muzecs: Embedded blocks for exploring computer science. In *Blocks and Beyond Workshop (Blocks and Beyond), 2015 IEEE*, pages 127–132, Oct 2015.

[3] Edison Robotic. https://meetedison.com.

[4] Evollve, Inc. Ozobot and OzoBlockly, 2016. http://ozoblockly.com/.

[5] Exploring Computer Science. ECS v7.0, 2016. http://www.exploringcs.org/curriculum.

[6] Firmata Project. Firmata firmware for arduino, v2.5.3. https://github.com/firmata/arduino.

[7] D. Franklin, C. Hill, H. A. Dwyer, A. K. Hansen, A. Iveland, and D. B. Harlow. Initialization in scratch: Seeking knowledge transfer. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 217–222, New York, NY, USA, 2016. ACM.

[8] Google. Chromebook. http://www.google.com/chromebook/.

[9] Google Blockly. http://developers.google.com/blockly.

[10] J. Margolis. *Stuck in the Shallow End: Education, Race, and Computing*. The MIT Press, 2008.

[11] J. Margolis, J. Goode, and G. Chapman. An equity lens for scaling: A critical juncture for exploring computer science. *ACM Inroads*, 6(3):58–66, Aug. 2015.

[12] M. Simon. Programming arduino. In *Programming Arduino*, pages 7–8, Oct 2012.

[13] H.-C. Steiner. Firmata: Towards making microcontrollers act like extensions of the computer. In *NIME*, pages 125–130, 2009.

[14] United States Bureau of Labor Statistics. http://www.bls.gov/ooh/computer-and-information-technology/home.htm.