Simplifying concurrency and monitoring on Arduino for Internet of Things

Ricardo Moran[1,2], Matías Teragni[1], Gonzalo Zabala[1]

[1]Universidad Abierta Interamericana, Centro de Altos Estudios en Tecnología Informática, Ciudad Autónoma de Buenos Aires, República Argentina

[2]Comisión de Investigaciones Científicas de la Provincia de Buenos Aires, Calle 526 e/ 10 y 11, La Plata, Buenos Aires, República Argentina

{Ricardo.Moran, Matias.Teragni, Gonzalo.Zabala}@uai.edu.ar

Abstract. The Internet of Things (IoT) presents several challenges and opportunities to improve people's lives. Experts agree on the importance of involving the community in the process of defining and creating IoT in order to succeed. Platforms like Arduino make it simple for non-technical people to build IoT devices. However, they also present difficulties that complicate their adoption and limit their reach. In this paper, we focus on the Arduino language and its limited support for concurrency and monitoring, which we deem essential for the IoT. We explore the existing solutions offered by the Arduino ecosystem and analyze their strengths and weaknesses. Finally, we propose an alternative solution based on a high-level programming language designed to tackle these issues with the help of an embedded virtual machine.

Keywords: Internet of Things, cloud computing, programming language, virtual machine, Arduino, concurrency, monitoring

1 Introduction

In the last decade, the development of networks of smart devices capable of sensing their environment, connect to the Internet, and publish data to cloud servers has led to a concept known as the Internet of Things (IoT). It is estimated that between 2008 and 2009 the number of "things" connected to the internet has surpassed the number of people [1]. While this evolution of the Internet architecture presents privacy and security risks [2] it also has the potential to improve people's lives [1].

In order to face these challenges in a satisfactory way several experts agree that it is important to reach out and involve more people in the IoT creation [3]. The Maker culture (a technology-based extension of the Do-It-Yourself movement [4]) could play an essential role in involving the society, in part thanks to the rise in popularity of platforms such as Arduino that make relatively easy and inexpensive to build IoT devices.

Arduino is a microcontroller board that has become one of the most popular platforms for building electronic projects, especially among hobbyists, artists, designers, and people just starting with electronics. Apart from being open-source hardware, one of the reasons for its popularity is its software library and integrated development environment (IDE) that provides an abstraction layer over the hardware details, making it possible to build interesting projects without a complete understanding of more advanced microcontroller concepts such as interrupts, ports, registers, timers, and such. At the same time, this abstraction layer can be bypassed to access advanced features if the user needs them making Arduino a platform suitable for both beginners and experts, allowing non-technical users to participate in the IoT phenomenon.

However, there are several aspects in which the Arduino language lacks proper support for the needs of an IoT project, introducing extra complexity that can overwhelm a beginner while building any kind of interesting system.

The first issue we identify is the lack of support for concurrency in the language itself. Even moderately complex problems require some sort of simultaneous task execution. Almost all IoT devices need to be able to perform some combination of the following tasks: communicate with other devices; read the value of several sensors; make decisions based on these values; connect to the Internet and publish their sensor data to the cloud [5]. Usually these tasks need to be done simultaneously but since the Arduino language does not provide any concurrency support it is left to the programmer to implement an arbitrarily complex scheduler in order to avoid interfering one task with another.

The second problem that Arduino presents is its complete lack for remote monitoring, a requirement for almost every IoT project. The lack of a standard protocol forces the programmer to design a different ad-hoc protocol for each project, which limits code reusability and is hard for non-technical users to implement properly. Other solutions are available in the form of open-source libraries but most of them suffer from issues that cannot be dismissed.

In this paper we will discuss the existing solutions, their limitations, and propose an alternative approach based on the use of a high-level programming language especially designed for these use cases that can be executed by a small virtual machine running in the microcontroller.

2 Related work

Regarding concurrency support, although the Arduino language does not provide any concurrency abstraction by default, third-party libraries are available that attempt to address this issue.

The most sophisticated libraries we surveyed are implementations of a real-time operating system for microcontrollers known as FreeRTOS [6] [7]. These libraries offer prioritized, preemptive multitasking. However, they are harder to use than the alternatives, requiring the user to specify several configuration parameters, which can be challenging for a novice programmer. Most libraries take a simpler approach, which is to support cooperative multitasking. These libraries differ in their implementation details and, thus, the programmer needs to understand the tradeoffs of each library in order to correctly predict the behavior of the program. Some libraries, like ArduinoProcessScheduler [8] and Task [9], force the user into the object-oriented paradigm and require the definition of classes for each independent process. Others, like Automaton [10] and Yet Another State Machine [11], are event-driven and focus on helping the user to create state machines. Most libraries, however, simply allow the user to specify which procedures should run concurrently by adding them to a global scheduler's list or using macros to define them. ArduinoThread [12] and everytime [13] take another approach worth mentioning, they do not implement truly independent tasks, but instead provide facilities to schedule the execution of procedures at desired intervals. Each time a task is executed it will run to completion. The main difference between the two is that everytime is fully based on compile-time macros, which makes it impossible to dynamically create new processes. While all these libraries and frameworks have their own tradeoffs, they all exhibit the same flaw: they require the user to understand the execution model provided by the library to use it effectively. If the programmer fails to understand the strengths and limitations of the library, they may be punished with potentially hard to debug errors [14]. In the case of the cooperative multitasking libraries, the limitation is evident: the user needs to be extra careful not to use blocking code (like the "delay" function or a long-running loop) or risk interfering with the execution of other tasks. This can be exacerbated by the fact that most third-party libraries, tutorials, and code examples on the web assume total control of the CPU and, thus, may use blocking functions that are not compatible with this programming style. For an experienced programmer this could represent a minor problem but for a beginner it could mean an unsurpassable challenge.

Alternative programming languages for the Arduino platform address the concurrency problem in different ways. We found several implementations of high-level languages and virtual machines for the Arduino platform. Most of them are based on preexisting general-purpose programming languages such as Java [15], Scheme [16] or Python [17]. In these cases, support for concurrency depends on the implementation and is tied to the mechanisms already existing in the language. A more noteworthy example is the Transterpreter project [18], a virtual machine explicitly designed to exploit concurrency on embedded systems. This virtual machine runs occam-pi programs on several platforms, one of which is Arduino [19]. Occam-pi is a variant of the occam programming language [20], especially designed to write concurrent programs based on communicating sequential processes (CSP) process algebra [21]. Occam-pi has a rich set of runtime libraries that provide functions for interacting with Arduino features such as the serial port, PWM and TWI. Regarding performance, the execution of bytecodes has been reported be 100 to 1000 times slower than the execution of native code.

Regarding monitoring and communication capabilities, the current options are limited. All Arduino boards have native support for serial communication and the Arduino language includes several functions to allow the user to read or write from the Serial. However, writing a custom protocol on top of the Serial can be challenging for beginners. For this reason, a popular choice for Arduino developers is the Firmata library [22].

Firmata is a standard protocol for communicating the Arduino with software on a host computer. It was designed to allow users to write custom firmware without having to create their own protocol. Client libraries for several languages are available online and the host implementation comes already bundled as one of the default libraries in the Arduino IDE. The protocol is extensible, and it can be used to communicate through the serial port, Wi-Fi, ethernet, and Bluetooth. The usefulness of Firmata has made it a very popular library but it has its drawbacks. On the one hand, the protocol details might not match exactly the requirements of the project, in which case the user is left with no alternative but to dive in the Firmata code and adapt it to suit its needs. This might be challenging for a beginner. On the other hand, if the protocol satisfies the project requirements then the simplest option is to upload the StandardFirmata that comes as an example with the library. The StandardFirmata was designed to include as much functionality as possible into a single firmware, but doing so leaves little room for any custom code the user might wish to add to the sketch (on Arduino UNO StandardFirmata uses 38% of program storage and 52% of dynamic memory) and adding new functionality to the StandardFirmata sketch can be complicated. Another option is to import Firmata into the sketch as a library, in which case the user would have to manually implement the functions to handle the relevant messages provided by the Firmata protocol.

A popular alternative is to use an IoT cloud solution. In this case, the user only needs to use a library provided by the platform it wishes to use and the library will take care of all the communication details. A survey of existing solutions in this space shows a variety of platforms offering services like real time data capture, data visualization and device management related tasks through remote cloud servers while implying “pay-as-you-go” notion [5]. While these platforms simplify the development of IoT systems, they require the user to write code to interact with their cloud services. In most cases, this code is relatively straightforward as it focuses on defining what information is sent to the cloud and what to do with the information received, but the behavior of those libraries is usually opaque and too complex, forcing the programmer to call periodically to certain functions that actually perform the synchronization. Of all the existing solutions we have focused on Thinger.io, the results of our analysis are shown in the sub-section below.

3 Proposed solution

We propose the implementation of a domain-specific language (DSL) supported by a virtual machine running on the Arduino. The nature of a DSL ensures that the logic usually present in IoT solutions can be expressed concisely and the abstraction layer it provides guarantees that the programmer cannot unintentionally interfere with the concurrent task execution. We call this language UziScript and we believe it would be a suitable alternative to the Arduino language for IoT.

4 Implementation

4.1 General architecture

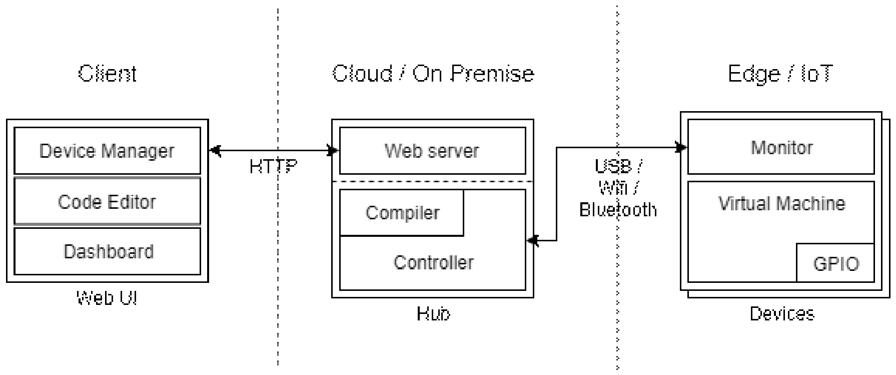

Fig 1 Architecture diagram of the UziScript toolchain and virtual machine

The architecture of the proposed solution consists of three distinct components: a firmware for the Arduino devices, a hub that handles the communication with the devices, and client tools that allow users to manage the entire system.

The firmware is a regular Arduino sketch that contains the language runtime, responsible for executing the programs as well as monitoring the state of the device. The hub, connected directly or indirectly to the Arduino boards, contains all the compilation tools and a web server that presents a REST API interface. The client tools are web applications that, through the hub server, allow to program, monitor and control all the devices in the system from any computer or phone that can reach the Hub server.

This architecture has several benefits.

1. Flexibility: the client tools, being a web app, could be used from any device, using a web browser or installing it as a native app; the hub could be deployed on the cloud or on premise, depending on the needs of the user.
2. Portability: the current firmware supports different Arduino boards (including UNO, Micro, Nano, MEGA 2560, and Yun) and we have also received reports of it working successfully on other compatible boards such as DuinoBot [23], Educabot [24], and TotemDUINO [25].
3. Interactivity: the runtime allows to compile and upload new programs at a fraction of the time required to compile an Arduino Sketch, this allows for a more interactive programming style that encourages experimentation and learning.
4. Scalability: a single hub could handle multiple devices and multiple hub instances could be deployed on a single system.

In the following sections we will explain some of the design choices we consider relevant for the IoT. For a detailed description of the implementation see [26].

Programming language. The UziScript programming language was designed to allow non-expert programmers to express concurrent tasks and facilitate monitoring the IoT device. Its syntax is based on C, which is familiar to most programmers including Arduino developers. In the following sections we will present some code examples.

To support concurrency, we added the "`task`" keyword, which represents behavior that can be executed periodically at a configurable rate. The scheduling and execution of each task is performed automatically by the runtime and the language allows the user to start, stop, pause, or resume any given task. Each task execution is independent.

To support monitoring, the runtime automatically keeps track of all the global variables in the program as well as the value of each pin in the Arduino. Thus, to publish some value to the outside world, the user only has to assign it to a global variable declared using the "`var`" keyword. To check and manipulate the value of either a digital or analog pin, the user does not have to write any code at all because they are monitored automatically.

Firmware. The firmware contains the virtual machine responsible for user program execution and a monitor program that allows it to interact with the hub controller. Periodically, this monitor program will send the status of the Arduino and receive commands, allowing the hub to fully control the virtual machine, including directly manipulating the variables and the pin state, debug the current user program, or download a new one.

Monitoring tools. The dashboard is a web tool that allows to monitor and control any global in the program or any pin on the Arduino device. In order to simplify this task for the user the dashboard allows to configure a blank canvas with a set of configurable widgets. Depending on the type of widget, it allows to either display or control a value.

5 Validation

To validate the effectiveness of the proposed approach we built an IoT greenhouse control system with the following requirements:

1. It can have one or more Arduino devices.
2. It should have a set of sensors to monitor the internal greenhouse conditions as well as a set of actuators that will allow to control said conditions.
3. It must have a dashboard that allows the user to check and control the entire system from a web browser.

We will compare three different implementations of the system. One will be written in the UziScript programming language. For the second we will use a project we built in 2017 using the Arduino language and the thinger.io cloud solution [27]. And for the third we will use the irrigation control system described in Donald Norris book "The Internet of Things: Do-It-Yourself at Home Projects for Arduino, Raspberry Pi and BeagleBone Black" [28].

Before we begin with the analysis it is important to note the differences between these three implementations. The following table shows the needed resources and functionality provided by each system.

	UziScript	Thinger.io	Norris
Microcontrollers required	1	1	2
Sensors	Temperature x2 Door open Ambient light	Temperature x2 Door open Ambient light	Moisture
Actuators	Fans Irrigation	Fans Irrigation	Irrigation

The entire code for the thinger.io is available at GitHub [27], Norris' code can be found in his book [28], and the UziScript program is displayed below.

```
import stepper from 'Stepper.uzi' {
     p0 = D3; p1 = D4; p2 = D5; p3 = D6; steps = 48;
}
import thermistor1 from 'Thermistor.uzi' { pin = A1; }
import thermistor2 from 'Thermistor.uzi' { pin = A2; }

var stepping; var temp1; var temp2; var n_readings = 10;

task control() running {
     "The stepping flag can be modified from the dashboard"
     if stepping { stepper.step(30); }
}

task sensors() running {
     var t1; var t2;
     repeat n_readings {
            t1 = t1 + thermistor1.readDegrees();
            t2 = t2 + thermistor2.readDegrees();
            delayMs(1);
     }
     temp1 = t1 / n_readings;
     temp2 = t2 / n_readings;
}
```

We will compare the following aspects of the solution: concurrency, monitoring, and code size:

Concurrency. The UziScript version uses concurrent tasks to group the different responsibilities of the program. This makes the code easier to understand and modify because it decouples the code to control the stepper motor from the code that checks the temperature sensor. In contrast, the sequential nature of the thinger.io version conflates the different tasks into one sequential procedure. The negative effects of this limitation can be seen in the code that gathers the temperature sensor readings on each tick. In Norris' version the separation of concerns into independent tasks is done using two Arduinos instead of just one. His implementation uses one Arduino for the control system and the other for the monitoring, and connects the two using a radio module. This provides true parallelism but makes the system more complex, expensive, and adds an extra point of failure, since the communication between the Arduinos can be interrupted.

Monitoring. While Norris and thinger.io versions have a large portion of the code related to the monitoring aspect of the solution, the UziScript version handles this almost with no extra code. Since the UziScript runtime handles the monitoring transparently, all the pins and global variables are automatically published to the host computer. This allows the user to see and change any value from the web dashboard without requiring extra code in the device to handle the communication.

Code size. Norris version uses two Arduino boards, so we need to consider both sketches (143 LOC + 18 LOC = 148 LOC). The thinger.io version takes advantage of several libraries provided by the cloud platform, which allowed us to write much less code (118 LOC). In contrast, the UziScript version can fit in a single page (23 LOC). Although we understand lines of code is not an optimal measure of the size of a program (especially when comparing different languages) the difference is worth noting and shows one of the benefits of using a DSL.

6 Conclusions and future work

In this paper we have analyzed the limited support for concurrency and monitoring in the Arduino language. We believe these issues limit the potential and reach of the Arduino regarding IoT.

We have explored some of the different solutions currently available as third-party libraries for the Arduino platform and we have proposed a solution based on a virtual machine and high-level language called UziScript.

We have described the implementation of said language, explaining how it solves the identified issues, and we validated its effectiveness by using it to build a small IoT system and comparing it to alternative solutions. The UziScript language presents benefits in code size, concurrency support, and monitoring capabilities. However, of all the implementations we have analyzed, the IoT cloud solution provides more functionality out of the box. This is not a surprise, being a mature commercial product, the cloud solution allows users to build custom dashboards using a simple web interface with support for multiple widgets for different sensors and actuators. The UziScript dashboard, on the other hand, offers limited visualization options and the virtual machine supports only a small number of sensors and actuators.

Despite its current limitations, we believe the implementation works as a successful proof of concept for the proposed architecture and we encourage anyone interested to proceed further with its development.

There are still a lot of improvements to be made. A major issue we found is performance. Our early measurements show the UziScript virtual machine to be 6 to 10 times slower than native Arduino code. While this performance might be good enough for some applications, we believe there is still room for improvement both in the runtime and compiler.

Another potential problem is the restriction on program size. Considering that most Arduino boards have limited memory, it is desirable that the virtual machine code and the user programs are as compact as possible.

Finally, we are working towards polishing the client tools to make them more suitable for the IoT. In particular, the dashboard should allow the user to use different visualization strategies to display the data from the devices.

Although it is not finished yet we believe the proposed approach is promising and we hope to explore it further in the future.

References

[1] D. Evans, The Internet of Things: How the Next Evolution of the Internet Is Changing Everything, Cisco Internet Business Solutions Group (IBSG), 2011.

[2] S. Sicari, A. Rizzardi, L. Grieco y A. Coen-Porisini, «Security, privacy and trust in Internet of Things: The road ahead,» Computer Networks, vol. 76, pp. 146-164, 2015.

[3] D. De Roeck, K. Slegers, J. Criel, M. Godon, L. Claeys, K. Kilpi y A. Jacobs, «I would DiYSE for it!: a manifesto for do-it-yourself internet-of-things creation,» NordiCHI '12 Proceedings of the 7th Nordic Conference on Human-Computer Interaction: Making Sense Through Design, pp. 170-179, 2012.

[4] J. Tanenbaum, A. Williams, A. Desjardins y K. Tanenbaum, «Democratizing technology: pleasure, utility and expressiveness in DIY and maker practice,» CHI '13 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 2603-2612, 2013.

[5] P. P. Ray, «A survey on Internet of Things architectures,» Journal of King Saud University - Computer and Information Sciences, vol. 30, nº 3, pp. 291-319, 2018.

[6] P. Stevens, «feilipu/Arduino_FreeRTOS_Library,» Github, [En línea]. Available: https://github.com/feilipu/Arduino_FreeRTOS_Library. [Último acceso: 13 12 2019].

[7] Floessie, «Floessie/frt,» Github, [En línea]. Available: https://github.com/Floessie/frt. [Último acceso: 13 12 2019].

[8] A. Wisner, «wizard97/ArduinoProcessScheduler: An Arduino object oriented process scheduler designed to replace them all,» 15 January 2017. [En línea]. Available: https://github.com/wizard97/ArduinoProcessScheduler. [Último acceso: 23 July 2017].

[9] M. Miller, «Makuna/Task,» Github, [En línea]. Available: https://github.com/Makuna/Task. [Último acceso: 13 12 2019].

[10] Tinkerspy, «tinkerspy/Automaton,» Github, [En línea]. Available: https://github.com/tinkerspy/Automaton. [Último acceso: 13 12 2019].

[11] bricofoy, «bricofoy/yasm/,» Github, [En línea]. Available: https://github.com/bricofoy/yasm/. [Último acceso: 13 12 2019].

[12] I. Seidel, «ivanseidel/ArduinoThread: A simple way to run Threads on Arduino,» 15 May 2017. [En línea]. Available: https://github.com/ivanseidel/ArduinoThread. [Último acceso: 23 July 2017].

[13] K. Fessel, «fesselk/everytime: A easy to use library for periodic code execution.,» 2 February 2017. [En línea]. Available: https://github.com/fesselk/everytime. [Último acceso: 23 July 2017].

[14] O. Meerbaum-Salant, M. Armoni y M. Ben-Ari, «Habits of programming in scratch,» ITiCSE '11 Proceedings of the 16th annual joint conference on Innovation and technology in computer science education, pp. 168-172, 2011.

[15] G. Bob, «HaikuVM: a small JAVA VM for microcontrollers,» 2017. [En línea]. Available: http://haiku-vm.sourceforge.net/. [Último acceso: 15 Junio 2017].

[16] R. Suchocki y S. Kalvala, «Microscheme: Functional programming for the Arduino,» de Scheme and Functional Programming Workshop, Washington, D.C., 2014.

[17] «PyMite - Python Wiki,» 2014. [En línea]. Available: https://wiki.python.org/moin/PyMite. [Último acceso: 15 Junio 2017].

[18] C. L. Jacobsen y M. C. Jadud, «The Transterpreter: A Transputer Interpreter,» Communicating Process Architectures 2004, vol. 62, pp. 182-196, 2004.

[19] C. L. Jacobsen, M. C. Jadud, O. Kilic y A. T. Sampson, «Concurrent event-driven programming in occam-π for the Arduino,» Concurrent Systems Engineering Series, vol. 68, pp. 177-193, 2011.

[20] M. Elizabeth y C. Hull, «Occam-A programming language for multiprocessor systems,» Computer Languages, vol. 12, nº 1, pp. 27-37, 1987.

[21] A. W. Roscoe y C. A. R. Hoare, «The laws of Occam programming,» Theoretical Computer Science, vol. 60, nº 2, pp. 177 - 229, 1988 .

[22] H.-C. Steiner, «Firmata: Towards Making Microcontrollers Act Like Extensions of the Computer,» NIME, pp. 125-130, 2009.

[23] A. Rojas, «Reporte Robótica Educativa,» Universidad Nacional de La Pampa (UNLPam), 2017.

[24] «Educabot,» [En línea]. Available: https://educabot.org/. [Último acceso: 13 12 2019].

[25] Totem, «TotemDUINO | Totemmaker.net,» Totemmaker.net, [En línea]. Available: https://totemmaker.net/product/totemduino-arduino/. [Último acceso: 13 12 2019].

[26] R. Moran, M. Teragni y G. Zabala, «A Concurrent Programming Language for Arduino and Educational Robotics,» de XXIII Congreso Argentino de Ciencias de la Computación (La Plata, 2017), 2017.

[27] CAETI GIRA, «GIRA/IoTGreenhouse,» Github, [En línea]. Available: https://github.com/GIRA/IoTGreenhouse. [Último acceso: 13 12 2019].

[28] D. Norris, The Internet of Things: Do-It-Yourself at Home Projects for Arduino, Raspberry Pi and BeagleBone Black, McGraw-Hill Education TAB, 2015.