Concurrencia en Arduino: Un enfoque basado en una máquina virtual

Ricardo Moran, Gonzalo Zabala, Matías Teragni

Centro de Altos Estudios en Tecnología Informática
Facultad de Tecnología Informática
Universidad Abierta Interamericana
Av. Montes de Oca 745, Ciudad Autónoma de Buenos Aires, República Argentina
(+54 11) 4301-5323; 4301-5240; 4301-5248
{Ricardo.Moran, Gonzalo.Zabala, Matias.Teragni}@uai.edu.ar

RESUMEN

Arduino es actualmente una de las plataformas más populares para robótica educativa debido a su bajo costo y gran cantidad de recursos disponibles en línea. Las librerías de Arduino proporcionan una capa de abstracción sobre los detalles del hardware, lo que permite construir proyectos interesantes aún sin tener experiencia en el tema. Sin embargo, su falta de soporte para la concurrencia dificulta algunos proyectos de robótica educativa. Como solución, hemos propuesto el uso de un lenguaje de programación concurrente soportado por una máquina virtual que se ejecuta en la placa Arduino. En este artículo, describimos la implementación de dicho lenguaje.

Palabras clave: robótica educativa, lenguaje de programación, máquina virtual, Arduino, concurrencia, programación, sincronización.

CONTEXTO

El presente proyecto está radicado en el Centro de Altos Estudios en Tecnología Informática (CAETI), dependiente de la Facultad de Tecnología Informática de la Universidad Abierta Interamericana. El mismo se encuentra inserto en la línea de investigación "Sociedad del conocimiento y Tecnologías aplicadas a la Educación". El financiamiento está brindado por la misma Universidad Abierta Interamericana.
Ricardo Moran percibe además una beca doctoral por parte de la Comisión de Investigaciones Científicas de la Provincia de Buenos Aires (CIC).

1. INTRODUCCIÓN

Arduino se ha convertido en una de las plataformas más populares para construir proyectos de electrónica, en especial entre hobbistas, artistas, diseñadores y principiantes en el área. Las librerías de software y el ambiente de desarrollo (IDE) de Arduino proveen una capa de abstracción sobre los detalles de hardware que hace posible construir proyectos interesantes sin una completa comprensión de conceptos más avanzados de microcontroladores como interrupciones, registros, y timers entre otros.
Al mismo tiempo, esta capa de abstracción puede ser ignorada para acceder a funcionalidades avanzadas si el programador lo requiere. Estas características hacen que la plataforma Arduino sea útil tanto para principiantes como para expertos.

Pero hay un aspecto en la que el lenguaje propuesto por Arduino carece de una abstracción adecuada, la concurrencia. Para todo proyecto que contenga cierta complejidad la estructura de setup() y loop() no es suficientemente expresiva. Incluso tareas moderadamente complejas requieren cierta ejecución de tareas simultáneas.
Particularmente, la mayoría de los proyectos de robótica educativa requieren la implementación de un dispositivo que realice dos o más tareas en simultáneo. Esto impone una limitación sobre el tipo de proyectos educativos que se pueden llevar a cabo, especialmente si el contenido que se desea

transmitir no está relacionado a robótica o programación en sí mismo.

2. LÍNEAS DE INVESTIGACIÓN, DESARROLLO E INNOVACIÓN

Propusimos la implementación de un lenguaje de programación concurrente soportado por una máquina virtual que se ejecuta en Arduino. Lo llamamos UziScript y tenemos la expectativa de que se convierta en el lenguaje al cual compilan entornos de programación visual como Physical Etoys, Scratch for Arduino y Ardublock , entre otros.

Dado que su propósito principal es educativo, fue diseñado en base a los siguientes principios:

1. Simplicidad: la máquina virtual debería ser fácil de entender, comprendiendo cómo realiza su trabajo.
2. Abstracción: el lenguaje debe proporcionar funciones de alto nivel que oculten algunos de los detalles relacionados con los conceptos de microcontroladores básicos y avanzados (como timers, interrupciones, concurrencia, pines de entrada/salida, etc.). Estos conceptos pueden introducirse posteriormente a un ritmo adecuado a las capacidades del estudiante.
3. Supervisión: debería ser posible monitorear el estado de la placa mientras esté conectada a la computadora.
4. Autonomía: los programas deben poder ejecutarse sin tener la placa conectada a la computadora.
5. Depuración: las herramientas deben proporcionar mecanismos para la detección de errores y la ejecución paso a paso del código. Sin herramientas de depuración, el proceso de corrección puede ser frustrante para un usuario sin experiencia.

3. RESULTADOS OBTENIDOS/ESPERADOS

Se diseñó e implementó un entorno de desarrollo integrado para robótica educativa cuyo objetivo es solucionar algunos problemas comunes que se observan en las herramientas actualmente disponibles en el mercado, particularmente la falta de soporte para expresar tareas concurrentes.

La solución tecnológica está compuesta por un firmware que se ejecuta en el robot y un conjunto de herramientas de desarrollo que pueden ejecutarse en una computadora o dispositivo móvil. El firmware desarrollado incluye un programa monitor encargado del protocolo de comunicación y una máquina virtual que abstrae los detalles del hardware. Se diseñó un lenguaje específico de dominio para robótica educativa y se implementaron dos interfaces de programación: una textual y otra visual (basada en programación por bloques). Ambas interfaces fueron diseñadas para usarse en simultáneo, proveyendo generación automática de una representación a partir de la otra.

Dado que la mayoría de los Arduinos contienen un solo microcontrolador, estos solo pueden ejecutar un hilo a la vez. Esto implica que todas las tareas definidas en el programa deben compartir un solo procesador. La máquina virtual, además de ejecutar las instrucciones del programa, es responsable de manejar la planificación de tareas. Esta decide qué tareas son ejecutadas, y cuando deben ser interrumpidas preventivamente. La estrategia de planificación es simple, la máquina virtual recorre la lista de tareas, encolando aquellas cuyo tiempo para correr haya sido alcanzado. Luego ejecuta cada instrucción de dichas tareas hasta que se realice una operación bloqueante, interrumpiendo la tarea actual y pasando a la siguiente. Cada tarea almacena entonces su contexto de ejecución (pila, contador de programa y frame pointer) permitiendo luego continuar la ejecución en el punto donde fué interrumpida. Algunas de las operaciones bloqueantes que fuerzan un cambio de contexto son la instrucción “yield”, todas las variantes de delay(), los saltos a instrucciones anteriores al puntero de ejecución actual, escribir el puerto serie cuando el buffer está lleno, leer del serie cuando el buffer está vacío, entre otras.

Poder simplemente expresar tareas concurrentes no es suficiente, dado que todas comparten los mismos recursos (memoria, pines, puerto serial) es necesario poder coordinarlas. Hemos implementado dos librerías externas que proveen mecanismos de sincronización. Ambas están escritas completamente en UziScript, permitiendo a los usuarios acceder a la lógica de sincronización de ser necesario sin la complejidad de enfrentarse a código de bajo nivel.

Este proyecto se encuentra en un estado de desarrollo avanzado y una primera versión fue publicada a principios de este año. Sin embargo, el lenguaje diseñado todavía presenta muchas limitaciones. Por ejemplo, el único tipo de dato soportado es punto flotante de 32 bits. Sería deseable soportar otros tipos de datos como integers, arrays, y strings.
El firmware también podría mejorarse. Quedan por implementar muchas primitivas de Arduino así como dar soporte a interrupciones de hardware. Estamos trabajando en un diseño que hará posible especificar tareas especiales que sólo se ejecutarán cuando ocurra una interrupción.
Finalmente, la performance también puede ser un problema. Nuestras mediciones iniciales muestran que UziScript es entre 20 y 60 veces más lento que el código Arduino nativo. Si bien esto puede ser suficiente para ciertas aplicaciones (sobre todo en el ámbito educativo), creemos que hay mucho lugar para optimizaciones.
Otro potencial problema es el tamaño de los programas. Considerando que muchas placas Arduino tienen una memoria reducida, es deseable que la máquina virtual y los programas del usuario sean lo más compactos posible. El conjunto de instrucciones, si bien fue diseñado para ser compacto, podría optimizarse. Y el compilador actual no realiza ninguna optimización adicional.

A pesar de las limitaciones descritas anteriormente, creemos que los beneficios que presenta la solución propuesta superan sus desventajas, especialmente en el contexto de robótica educativa.

4. FORMACIÓN DE RECURSOS HUMANOS

El equipo de trabajo está conformado por un investigador adjunto del Centro de Altos Estudios en Tecnología Informática (CAETI) quien ejerce el rol de director del proyecto, dos doctorandos, y un ayudante alumno de la Facultad de Tecnología Informática de la Universidad Abierta Interamericana.

5. BIBLIOGRAFÍA

- Arduino, "Arduino Playground - Structure," [Online]. Available: http://playground.arduino.cc/ArduinoNotebookTraduccion/Structure. [Accessed 23 Julio 2017].
- M. Resnick, "MultiLogo: A Study of Children and Concurrent Programming," Interactive Learning Environments, vol. 1, no. 3, 1990.
- R. Moran, M. Teragni and G. Zabala, "A Concurrent Programming Language for Arduino and Educational Robotics," in XXIII Congreso Argentino de Ciencias de la Computación (La Plata, 2017), 2017.
- F. Oudert, "fabriceo/SCoop: Simple Cooperative Scheduler for Arduino and Teensy ARM and AVR," 13 January 2013. [Online]. Available: https://github.com/fabriceo/SCoop. [Accessed 23 July 2017].
- A. Wisner, "wizard97/ ArduinoProcessScheduler: An Arduino object oriented process scheduler designed to replace them all," 15 January 2017. [Online]. Available: https://github.com/wizard97/ArduinoProcessScheduler. [Accessed 23 July 2017].
- I. Seidel, "ivanseidel/ArduinoThread: A simple way to run Threads on Arduino," 15 May 2017. [Online]. Available: https://github.com/ivanseidel/ArduinoThread. [Accessed 23 July 2017].
- K. Fessel, "fesselk/everytime: A easy to use library for periodic code execution.," 2

February 2017. [Online]. Available: https://github.com/fesselk/everytime. [Accessed 23 July 2017].
- G. Bob, "HaikuVM: a small JAVA VM for microcontrollers," 2017. [Online]. Available: http://haiku-vm.sourceforge.net/. [Accessed 15 Junio 2017].
- R. Suchocki and S. Kalvala, "Microscheme: Functional programming for the Arduino," in Scheme and Functional Programming Workshop, Washington, D.C., 2014.
- "PyMite - Python Wiki," 2014. [Online]. Available: https://wiki.python.org/moin/PyMite. [Accessed 15 Junio 2017].
- C. L. Jacobsen and M. C. Jadud, "The Transterpreter: A Transputer Interpreter," Communicating Process Architectures 2004, vol. 62, pp. 182-196, 2004.
- C. L. Jacobsen, M. C. Jadud, O. Kilic and A. T. Sampson, "Concurrent event-driven programming in occam-π for the Arduino," Concurrent Systems Engineering Series, vol. 68, pp. 177-193, 2011.
- M. Elizabeth and C. Hull, "Occam-A programming language for multiprocessor systems," Computer Languages, vol. 12, no. 1, pp. 27-37, 1987.
- A. W. Roscoe and C. A. R. Hoare, "The laws of Occam programming," Theoretical Computer Science, vol. 60, no. 2, pp. 177 - 229, 1988 .
- G. R. Andrews and F. B. Schneider, "Concepts and Notations for Concurrent Programming," ACM Computing Surveys (CSUR), vol. 15, no. 1, pp. 3-43, March 1983.
- Grupo de Investigación en Robótica Autónoma del CAETI (GIRA), "Physical Etoys," 2010. [Online]. Available: http://tecnodacta.com.ar/gira/projects/physical-etoys/. [Accessed 15 Junio 2017].
- Citilab, "About S4A," 2015. [Online]. Available: http://s4a.cat/. [Accessed 15 Junio 2017].
- "Ardublock | A Graphical Programming Language for Arduino," [Online]. Available: http://blog.ardublock.com/. [Accessed 15 Junio 2017].
- J. Smith and R. Nair, Virtual Machines: Versatile Platforms for Systems and Processes, San Francisco, CA: Morgan Kaufmann Publishers Inc., 2005.
- T. Luerweg, "Stack based programming paradigms," in Seminar Concepts of Programming Languages (CoPL), 2015.
- D. Ingalls, T. Kaehler, J. Maloney, S. Wallace and A. Kay, "Back to the future: the story of Squeak, a practical Smalltalk written in itself," in Proceedings of the 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, 1997.
- A. Goldberg and D. Robson, Smalltalk-80: The Language and its Implementation, Addison-Wesley Longman Publishing Co., 1983.
- Google, "Google for Education > Blockly," [Online]. Available: https://developers.google.com/blockly/. [Accessed 26 4 2018].
- C. A. R. Hoare, "Communicating sequential processes," Communications of the ACM, vol. 21, no. 8, pp. 666-677, 1978.
- J. Meyerson, "The Go Programming Language," IEEE Software, vol. 31, no. 5, 2014.
- N. Togashi and V. Klyuev, "Concurrency in Go and Java: Performance analysis," in 4th IEEE International Conference on Information Science and Technology (ICIST), 2014.