

IoT Software by Dataflow Programming in Mruby Programming Environment

Kazuaki Tanaka(✉), Chihiro Tsujino, and Hiroyuki Maeda

Kyushu Institute of Technology, Fukuoka 820-8502, Japan
kazuaki@mse.kyutech.ac.jp

Abstract. In IoT software development, a method to implement software by focusing on data flow has been proposed, which is called dataflow programming. On the other hand, sensor devices are generally implemented in microcontrollers with limited resources for compactness, power savings and costs, so sensor devices are not suitable for executing data flow programs.

In this paper, an environment running the scripting language mruby is described to execute a dataflow program on a small microcontroller. The execution of the data flow program is performed asynchronously by multiple nodes, which handle the sensor data. The execution environment of a dataflow program must support this style of execution. Since mruby can execute multiple programs concurrently, it is well suited to implement dataflow programs. By generating mruby code from a program developed in Node-RED, one of the popular dataflow programming environments, the mruby program is executed on a single-chip microcontroller, which includes mruby virtual machine.

Keywords: IoT · Dataflow programming · Node-RED · Mruby

1 Introduction

One of the challenges in realizing an IoT-based system is the implementation of a series of data processing from data input to output. The IoT system is realized through the procedure of sending the data obtained from the IoT device to the cloud via the network. It is complex to implement each procedure in programs and to define the interface between them.

Dataflow programming has been proposed as a means of adequately describing the flow of such sensor data. Dataflow programming is an effective programming tool to implement IoT systems.

Open source software such as Node-RED has been released and adopted for dataflow description in the cloud [4]. However, Node-RED is implemented in JavaScript, which requires much resources not only during development but also during execution. In addition, because the dataflow implemented in JavaScript is executed sequentially by an interpreter, it becomes a dynamic execution environment and cannot be installed in a small microcontroller.

In this paper, a development environment and an execution environment are proposed to solve this problem.

© Springer Nature Switzerland AG 2020
O. Gervasi et al. (Eds.): ICCSA 2020, LNCS 12252, pp. 212–220, 2020.
https://doi.org/10.1007/978-3-030-58811-3_15

2 Mruby

Mruby is a programming language optimized to execute programs on embedded systems, based on the object-oriented programming language Ruby. Ruby is widely used as a standard development language for web applications because of its high readability, maintainability and development efficiency. Mruby was released as open source software in 2012 and is used in embedded systems such as network routers, consumer games, and so on [1,2].

2.1 Mruby Execution Environment

Mruby consists of a mruby compiler and a mruby virtual machine (mrubyVM). The mruby compiler compiles the mruby source code and generates mruby bytecode. The mruby bytecode is a device-independent and intermediate binary code that runs on mrubyVM (Fig. 1). Since mrubyVM requires less memory at execution time, it can be executed on small microcontrollers. The smallest mrubyVM configuration executes a program within 20 KB of memory.

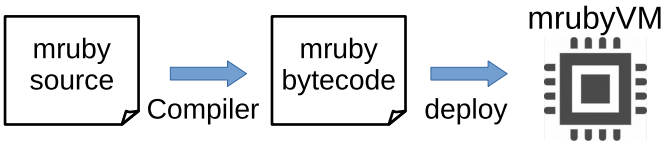

Fig. 1. mruby compiler and mrubyVM

The mruby compiler compiles mruby source code and generates bytecode. Bytecode is device-independent, and you can run the mruby program in the target device simply by porting mrubyVM. Porting mrubyVM to the target device is easy since mrubyVM is implemented in C99 standard.

2.2 Mruby/c

Mruby was released in 2012 and is now being used in a variety of commercial products. The most frequent feedback from mruby application developers was about execution on smaller microcontrollers. To achieve this goal, we developed mruby/c as another implementation derived from mrubyVM. While mruby provides a mruby compiler and mrubyVM, mruby/c only provides mrubyVM. That is, mruby and mruby/c execute the exactly same bytecodes. It is designed to support execution on small microprocessors, so it consumes less memory when running mruby/c. Mruby/c can execute the meuby bytecode within about 1/8 of the memory that mruby requires.

Also, mruby/c targets small microcontrollers (e.g., single-chip microcontrollers) that do not have an OS. For this reason, mruby/c provides memory management (dynamic memory allocation), task management, and file system

functions that are usually provided by the OS. However, these features are the minimum configuration required to run mruby/c. As a result, mruby/c can be ported to a variety of microcontrollers, regardless of OS or CPU architecture. Mruby/c supports PIC32, ARM Cortex-M, RISC-V and ESP32 by default, as well as x64 and x32.

2.3 Concurrent Execution

Mruby/c can also execute multiple programs concurrently [3]. Small microcontrollers may not support an operating system due to processor performance and memory limitations. However, developers wish to use the feature of the OS to process multiple programs in parallel (e.g., tasks and processes).

Mruby/c has a task control block (TCB) for concurrent execution. The mruby compiler compiles each program that should execute concurrently and generates mruby bytecode. Bytecode is a sequence of intermediate code, and the VM fetches instruction words from bytecode and executes them sequentially. The TCB holds the execution state of the program and the next execution point in the bytecode. Since the VM executes byte code based on TCB information, concurrent execution of multiple programs can be realized by switching multiple TCBs (Fig. 2).

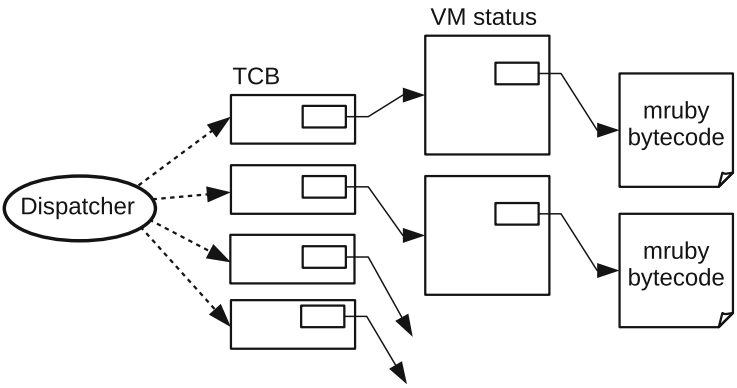

Fig. 2. TCB

The VM dispatches TCBs at fixed time intervals (1[ms] by default). The TCB is associated with a VM status that holds the execution state, such as the execution position of the executing bytecode. When the VM switches the TCB, the running bytecode is switched, allowing multiple programs to be executed concurrently. Note that when running a single program in mruby/c, TCB is not needed, so the mruby/c VM refers directly to the VM status.

3 Implementing a Dataflow Program in mruby

A dataflow program consists of nodes that process data and wires that connect the nodes to each other. Figure 3 shows the programming environment of Node-RED. Nodes are divided into those that generate data, those that process data, and those that output data. The sensor is considered to be the node that generates the data and the data transmission to the cloud is considered to be the node that outputs the data. A wire connects the nodes.

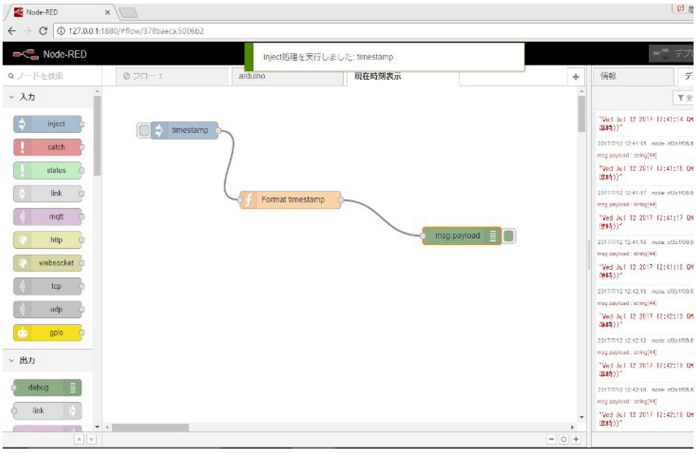

Fig. 3. Node-RED, a dataflow programing environment. Some nodes and wires are placed.

The processing of each node is executed asynchronously. dataflow programs, unlike procedural programs, cannot specify the order of execution. All nodes are triggered to start execution, although the execution of the node is different depending on the type of node. When a node's processing is complete, it generates and outputs data. The output data is sent to a node in the subsequent node connected by a wire, and the processing of that node begins.

Since the execution time of a node is non-deterministic, the order of execution cannot be determined in a prior. Therefore, the nodes must be executed asynchronously. For example, in the case of a dataflow program like the one shown in the Fig. 4, the completion time of node 1 and node 2 is uncertain, so the timing of data being passed to the wire is also uncertain. Therefore, node 3 starts execution, triggered by the completion of execution of node 1 or node 2.

3.1 Wires

The wires that connect the nodes are implemented by the data structure of the queue. For a simple one-to-one wire that connects nodes, it is enough to

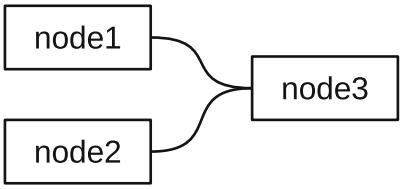

Fig. 4. Simple nodes and wires. This case contains three nodes and two wires. Output data from node 1 and node 2 will flow into node 3 through wires.

trigger the completion of a node to start the execution of a subsequent node. However, when connecting nodes to many-to-many, a simple trigger cannot be implemented, because it cannot determine the order of the triggers. Therefore, we implement the wires that connect the nodes in an abstraction queue (Fig. 5). With each node holding a queue, the flow of data can be implemented as an input of data into the queue.

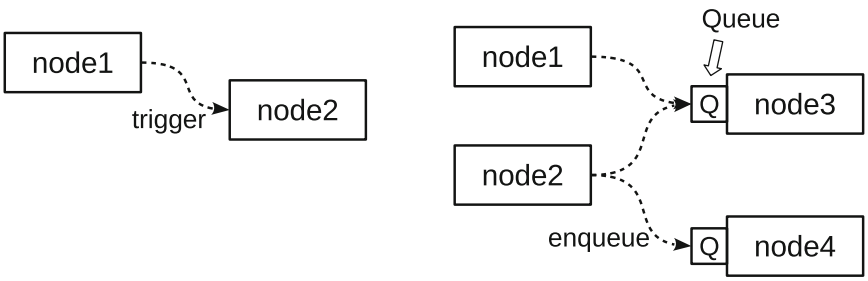

Fig. 5. Implementation of wires. (left) one-to-one wire between nodes, using trigger (right) many-to-many wire between nodes, using queue

Since data input to the queue (enqueue) is done asynchronously, exclusion processes are necessary. Mruby provides a Mutex class for exclusion processing. Mutual exclusions can be implemented by simply protecting operations on the queue with a Mutex instance, while keeping concurrency intact.

3.2 Nodes

Since node processing must be executed asynchronously, all nodes are implemented as programs that are executed concurrently. The wires connecting the nodes are implemented as queues that can support concurrent processing, so the execution and output of the nodes can be executed asynchronously (i.e., independently). The node receives input from the previous node by queue. Therefore, the processing of each node is implemented as Listing 1.1.

Listing 1.1. Generated mruby source code from node2

```
while true
  if $Q_node2.size == 0 then
    sleep 0.005
    next
  end
  # Enqueue
  data = $Q_node2.shift
  #
  # procedure in this node
  #
end
```

The Listing 1.1 shows an example of the mruby source code generated from a node named `node2`. Since multiple programs can be executed concurrently, the processing of nodes can be implemented as a permanent loop. If there is no data in the queue (`$Q_node2`) associated with this node, processing of the node is not necessary. In order to keep fairness of concurrent execution through multiple programs, the small period sleep is executed. The sleep in mrubyVM is pass the execution period to other programs for smooth concurrent execution. On the other hand, if there is data in the queue, it is taken out of the queue (dequeue) and processed in the node.

3.3 Automatic Generation of mruby/c Programs

In this study, the development environment for the dataflow program is Node-RED. mruby/c program is generated from the node and wire information created in Node-RED (Fig. 6). Node-RED is both a development environment and a program execution environment. However, it is not possible to run Node-RED on small microcomputers because of the CPU power and small on-board memory. To solve this problem, a mechanism to convert Node-RED dataflow programs into multiple mruby programs (mruby/c dataflow program generator) has been developed.

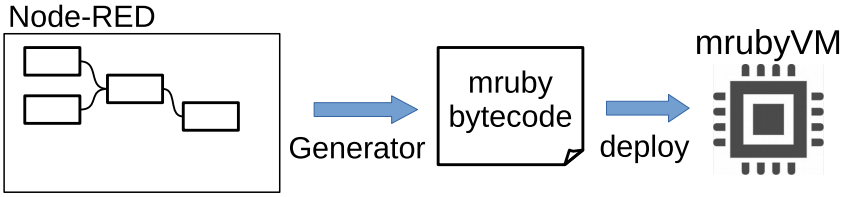

Fig. 6. deploy mruby program into targetdevice, by using mruby/c dataflow program generator

As described above, the nodes and wires of the dataflow program can be implemented by several mruby programs and queues. The dataflow programs developed in Node-RED (i.e., node and wire information) can be exported as JSON format files. The mruby/c dataflow program generator parses this file and generates a program corresponding to the node and an initialization program for the queue corresponding to the wire.

4 Execution Environment and Results

To test the mruby/c dataflow program generator, we generated mruby bytecode from a Node-RED dataflow program and ran it on a microcontroller. The microcontroller board RBoard executes the mruby bytecode. The RBoard has one-chip microcontroller, CY8C5888AXI-LP096 (Cypress Semiconductor, ARM Cortex-M3) (Fig. 7). RBoard is a general-purpose microcontroller board commercially available at Shimane Information Processing Center, and its onboard memory is 64 KB, and it is a microcontroller that assumes an execution environment without an OS.

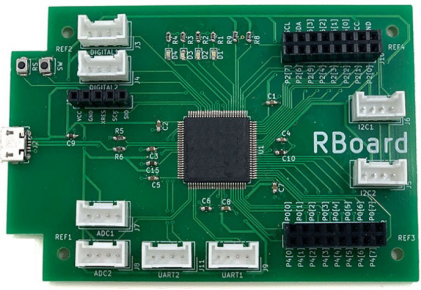

Fig. 7. RBoard; microcontroller with mruby/c VM. ARM Cortex-M3 with 64 KB RAM/256 KB flash memory, GPIO, USB and Grove system interface (by Speed) are equipped.

4.1 Mruby Bytecode Generation

The program using the input node from the temperature sensor and the output node to the console is shown in Fig. 8.

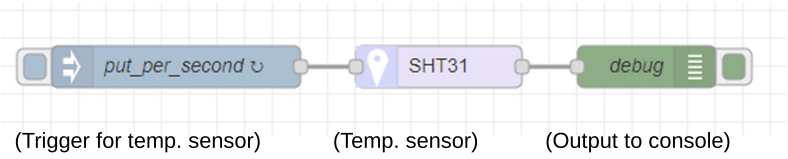

Fig. 8. Example of sensing temperature. Nodes for trigger, temperature sensor and console output are placed.

Three nodes and two wires are implemented in the data flow program shown in Fig. 8. Node-RED programs can be written using a drawing editor, which can also be output as a JSON file. The mruby/c dataflow program generator generates multiple mruby source codes from JSON files.

4.2 Bytecode Execution

Transfer multiple mruby bytecodes generated by the mruby/c dataflow program generator to the microcontroller board. Since RBoard has a USB-serial interface, it can be transferred directly from the development environment (the environment for developing Node-RED dataflow programs). The transferred mruby bytecodes will start execution in the microcontroller board.

At the start of execution, all the queues, which are wires in the dataflow program, are empty, so the program is in a suspended state. The first node in the dataflow, which is called the data source, generates the data and puts the data into the queue. Since the queue for the following node is no longer empty, the program for this node will be executed. After this, the execution propagates in the same way.

The last node outputs the data to the external. In this experiment, the data is output to the console. The last node does not generate any data, and when the processing at all nodes is finished, all queues are emptied, and the dataflow program goes into a suspended state.

4.3 Result of an Experiment

A data flow program including conditional branching is shown in Fig. 9. This data flow program obtains the value of the temperature sensor and flashes four LEDs according to the value of the temperature.

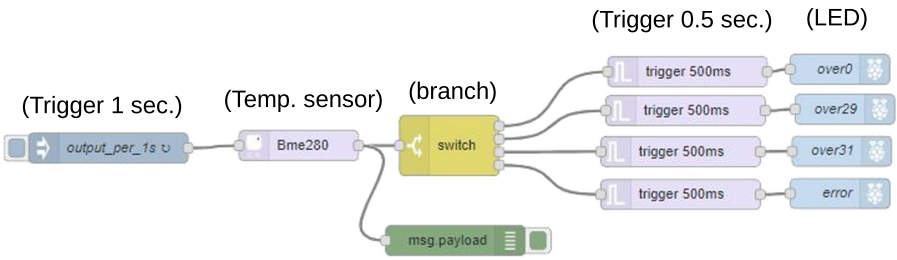

Fig. 9. Example of sensing temperature and blinking LEDs.

To ensure that the program is running asynchronously, a temperature sensor value is obtained every 1 s and the LED is blinking every 0.5 s. The mruby data flow generator also generates an mruby case statement (which is equivalent to a C switch statement) from the branch node of the multi-conditions.

Figure 9 is a data flow program consisting of 12 nodes and 11 wires. This mruby data flow generator generates 13 mruby programs. They are 12 nodes and one program to initialize the queue. All these programs were transferred to the RBoard microcontroller board and executed. As a result of the experiment, it was confirmed that the LED corresponding to the value obtained from the temperature sensor blinked.

5 Conclusion

In this paper, we describe a mechanism for executing dataflow programming for IoT software development on a small microcontroller. The execution of a dataflow program requires a lot of resources at runtime, so it is not suitable for small microcontrollers. We solved this problem by generating the mruby bytecode using the connection information of the nodes and wires in the dataflow. The mrubyVM, which runs mruby bytecode, is small enough to run on a single-chip microcontroller and supports concurrent execution of multiple programs, making it suitable for implementing dataflow program.

As IoT applications become more widespread in society, the weight of the data flow programming environment in IoT software development is expected to increase. There is also a need to combine traditional procedural programs and use data flow programs to build IoT systems. Mruby is able to remove the barriers between these two programming environments. Mruby, and the mruby bytecode generator developed in this study, are open source software[5,6]. We hope that mruby can contribute to solving this problem. The source code of this work will release at https://github.com/mruby-lab.

References

1. Maeda, H., Tanaka, K.: Hardware acceleration of language processing in scripting programming languages. In: Misra, S., et al. (eds.) ICCSA 2019. LNCS, vol. 11621, pp. 407–416. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24302-9_29
2. Nagumanthri, A.D., Tanaka, K.: Internet of Things with mruby. In: International Conference on Information Technology, Next Generation Information Technology Summit (InCITe-2016) (2016)
3. Tanaka, K., Maeda, H., Higashi, H.: Concurrent execution in scripting programming language 'mruby'. In: Gervasi, O., et al. (eds.) ICCSA 2018. LNCS, vol. 10962, pp. 136–146. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-95168-3_9
4. Node-RED. https://nodered.org/
5. mruby. https://github.com/mruby/mruby
6. mruby/c. https://github.com/mrubyc/mrubyc