

Data-Flow Visual Programming Environment for Small IoT Devices

Sota Ogura[1], Kazuaki Tanaka[1(✉)], Ko-ichiro Sugiyama[2], and Miu Kawahara[2]

[1] Kyushu Institute of Technology, Kitakyushu, Japan
kazuaki@ics.kyutech.ac.jp
[2] National Institute of Technology, Matsue College, Matsue, Japan

Abstract. Embedded systems, such as smartphones, home appliances, and automotive cars, are widely used in various settings. Furthermore, demand for embedded systems continues to rise, driven by factors such as the aging population, declining birthrate, and the need to reduce labor costs in factories and companies.

To address the shortage of C and C++ programmers and enhance the accessibility of embedded systems development, we propose a method that utilizes Node-RED, a visual programming tool. And then mruby/c, a language that combines the benefits of Ruby and C/C++, and enable even programming novices to develop embedded systems. We also incorporate Docker Compose to simplify the process of building a development environment, and release as a open source software.

Our approach offers a promising solution for engineers seeking to improve their embedded systems development skills or those who want to enhance the productivity and readability of their code. By leveraging Node-RED and mruby/c, this study provides an innovative solution that can enable a wider range of developers to participate in embedded systems development, ultimately contributing to the growth of this field.

Keywords: mruby/c · IoT · Node-RED · Programming Environment

1 Introduction

1.1 Background

These days, embedded systems are used in various places and situations, and their demand is increasing year after year. Factors such as a declining birth rate, an aging population, and cost reduction in companies and factories are contributing to this rise in demand. For embedded systems, development in C is preferred because of its immediacy. As a result, there is a need for an embedded development environment that is easily used for beginners who have never used C before.

In recent years, the importance of utilizing data such as IoT and AI as critical technological elements are highlighted. To satisfy this demand, we focused on data-flow visual programming tools, which are good for understanding data flow

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
O. Gervasi et al. (Eds.): ICCSA 2023 Workshops, LNCS 14107, pp. 150–161, 2023.
https://doi.org/10.1007/978-3-031-37114-1_11

when developing IoT applications. In this study, we propose a method for building an embedded development environment that enables a wider range of users to develop embedded systems using Node-RED, a data-flow visual programming tool, and mruby/c, which shares the productivity and readability characteristics of Ruby compared to C.

1.2 Objective

The objective of this research is to develop a programming environment for small devices that allows users to intuitively comprehend the flow of data between devices for IoT development. To provide an environment that enables a wider range of users to develop IoT applications, we use a data-flow type visual programming language, which is easy to comprehend. Developers can implement application programs by visually placing nodes and wiring among them. The programs for microcontrollers are automatically generated from these visual programs. This approach allows users to develop IoT applications with ease and enhances development efficiency.

2 Application Configurations

Table 1 shows the development environment used in this work. We choose the programming language Ruby, which offers better readability and productivity than C. The data-flow visual programming tool we choose is the Node-RED environment. All these environments are coded as a docker-compose copen-source software.

Table 1. Development Environment

OS	Windows 11
Chrome browser	>89
Node.js	> ver.16.17.1
Node-RED	> ver.3.0.2
mruby/c IDE	> ver.1.03
Docker Compose	> ver.3.7

2.1 mruby/c

murby/c is a language for embedded development that inherits the high productivity and readability characteristics of Ruby, making it easy to use for beginning programmers [4]. Furthermore, its compiler can convert Ruby code to byte-code that can be executed on a VM (Virtual Machine), reducing memory consumption during program execution. This feature allows for development on microcontrollers with limited computing resources. If a low-power microcontroller chip is used, it can be implemented in a power-saving manner. In this study, we aim to construct a programming environment using the Ruby language by introducing mruby/c.

2.2 Node-RED

This study leverages Node-RED [5], a data-flow visual programming tool developed by IBM to interconnect hardware devices, APIs, online services, etc. Node-RED is built on Node.js to leverage an event-driven, non-blocking model. It can also run on low-cost hardware, such as a Raspberry Pi at the edge of the network, as well as in the cloud.

The programming method is to create a flow by arranging blocks, called nodes, each of which is given a function, and connecting them with wires. Nodes send and receive data to and from each other, and perform various processes (Fig. 1). This programming method is more visually represented in the editor, making it an easy-to-use tool for a wide range of users.

Node-RED also allows users to create their own nodes. Node design and layout of configuration items are done in HTML, and the processing within nodes is done in JavaScript.

Nodes process external events such as messages and GPIO hardware changes from the forward node to which they are basically connected and send them out to the backward node. Each node has one input port and multiple output ports as needed. There are also three types of nodes implemented.

Input Nodes:	Has only output ports.
	Flow can be started manually or at regular intervals.
Output Nodes:	Has only an input port.
	Displays messages in the sidebar in the editor.
Functional Nodes:	Has both input and output ports.
	Performs conditional branching and various other processes.

In this study, we developed a node for microcontrollers using this functionality. The details are described in Sect. 3.2.

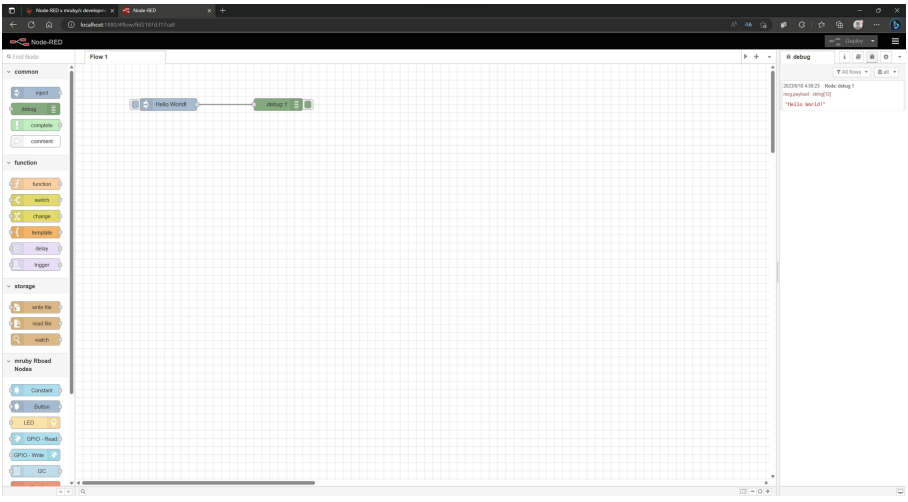


Fig. 1. Node-RED Development Environment

2.3 Docker Compose

Docker is a containerized virtualization technology provided by Docker Inc. Container-based virtualization technology is a technology that creates an execution environment in which a middleware environment called a container is built from the Docker Engine, a resident program for using Docker running on the host OS, without booting a guest OS. One advantage is that applications can run more lightly because the load on the computer is reduced by using only the minimum amount of memory and CPU required. Another advantage is resource efficiency, since an OS is not required for each application. Even complex architectures can be easily shared with a Dockerfile and the necessary files. An image of container-based virtualization is shown in Fig. 2.

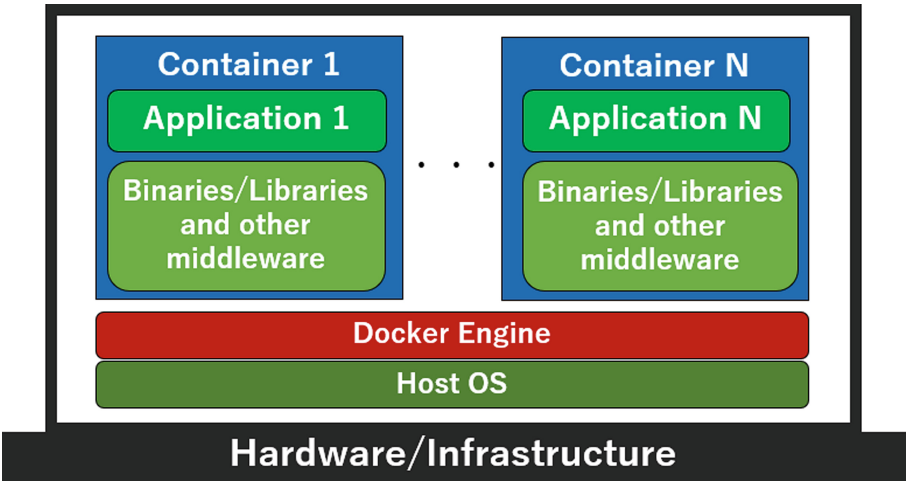


Fig. 2. Image Figure of Container-Based Virtualization Technology

However, to start a container, it is necessary to code complex commands with options, which can easily lead to mistakes and prevent smooth environment sharing. Docker Compose can automate this process by allowing you to define web servers, DB servers, etc. in the 'docker-compose.yml' file, and then start and configure all the containers needed to run them at once (Fig. 3).

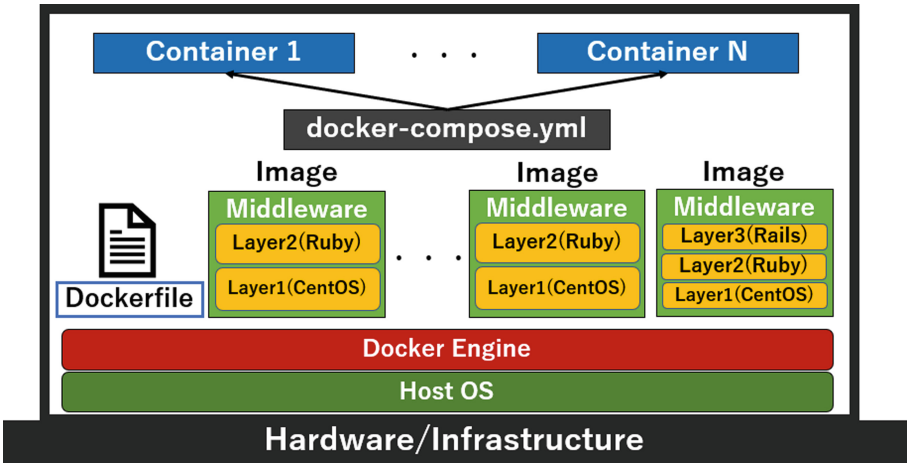


Fig. 3. Image Figure of Docker Compose

In this study, the Node-RED execution environment was automated with Docker and the container started with Docker Compose. The docker-compose.yml file created for the build is shown in the code 1.1. Each item is described below. The following describes the “Image Creation" and “Container Start" necessary to start Docker.

version: Specify the version of Docker Compose
services: Describe the service container to be started
build: Create an image from the dockerfile in the current directory
ports: Release Container number 1880
volumes: Bind-mount the current directory to the /data directory

Code 1.1. docker-compose.yml

```
version: "3.7"

services:
  node-red:
    build: ./node-red
    ports:
      - "1880:1880"
    volumes:
      - node-red-data:/data

  sinatra:
    build: ./sinatra
    ports:
# - 4567:4567
      - 80:4567
volumes:
  node-red-data:
```

Creationg an Image. An image is a collection of configuration files necessary to start a container, the application's execution environment. The same execution environment can be run on various machines by sharing this image. The image is created by making a file named "Dockerfile" and describing the settings involved in building the image in the file. The creation should describe the following. In addition, the dockerfile created in this study is shown in Code1.2.

FROM: Describe the image name and tag name based on which image you want to create.
COPY: Describes the names of files and directories to be copied into the container.
RUN: Execute commands to install middleware.

Code 1.2. Dockerfile

```
FROM nodered/node-red:latest


COPY node-red-contrib-mrubyc-rboard ./node-red-contrib-mrubyc-rboard

RUN npm install ./node-red-contrib-mrubyc-rboard

#common
RUN rm ./node_modules/@node-red/nodes/core/common/05-junction.html
RUN rm ./node_modules/@node-red/nodes/core/common/05-junction.js
RUN rm ./node_modules/@node-red/nodes/core/common/25-catch.html
RUN rm ./node_modules/@node-red/nodes/core/common/25-catch.js
RUN rm ./node_modules/@node-red/nodes/core/common/25-status.html
RUN rm ./node_modules/@node-red/nodes/core/common/25-status.js

RUN rm ./node_modules/@node-red/nodes/core/common/60-link.html
RUN rm ./node_modules/@node-red/nodes/core/common/60-link.js

RUN rm ./node_modules/@node-red/nodes/core/common/98-unknown.html
RUN rm ./node_modules/@node-red/nodes/core/common/98-unknown.js
```

Container Start. A container is a process isolated from other processes running on the OS. When it is launched, the Docker daemon first creates a process name space dedicated to the container and then executes the processes contained in the container as members of that name space.

Type "docker-compose up -d" when starting containers at the command prompt. This will start a series of containers in detached mode. By sharing the docker-compose.yml file and the dockerfile, the same development environment can be built on other machines.

3 Method

3.1 System Configurations

The system configuration to be developed in this study is shown in Fig. 4. The user creates a flow by connecting nodes on Node-RED. The JSON code stored in the created flow is extracted. Then, based on the JSON code, the Ruby code to run on the microcontroller board and convert it from the converted Ruby code to byte code. RBoard executes the program with the received byte-code.

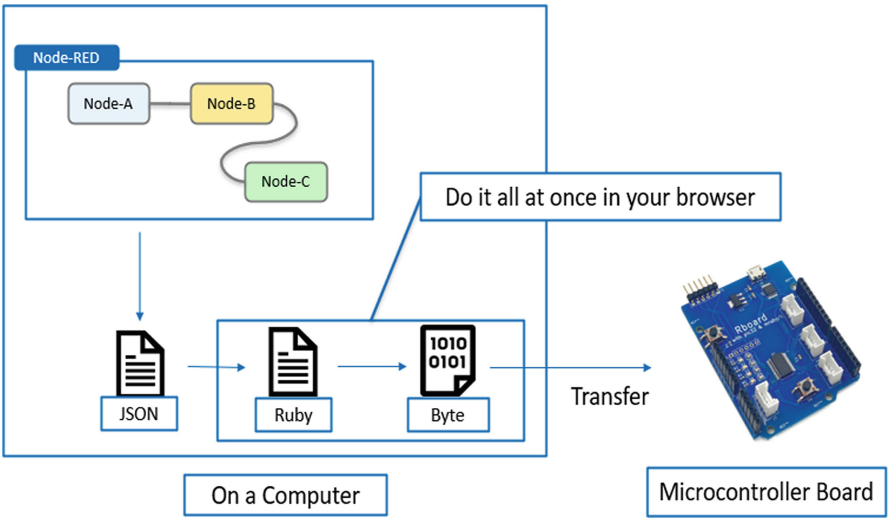


Fig. 4. System Configurations

3.2 Node Development

Node-RED allows you to create original nodes. The created node and the edit dialog for the LED node are shown in Fig. 5. Creating a node, the appearance of the node, the design of the edit dialog, and the configuration items of the node, which require JavaScript and HTML, are reflected on Node-RED by copying the file containing the node program with the COPY command and installing it with the RUN command. The reflected node is displayed as shown in the left figure of Fig. 5, and can be placed in the editor on Node-RED by drag and drop to create a flow. Nodes placed in the editor can be configured by simply selecting functions and entering numerical values in the edit dialog as shown in the center figure. The function-ruby node shown on the right can be freely written in Ruby, so it is used when you want to perform processing that is difficult to do by arranging nodes.

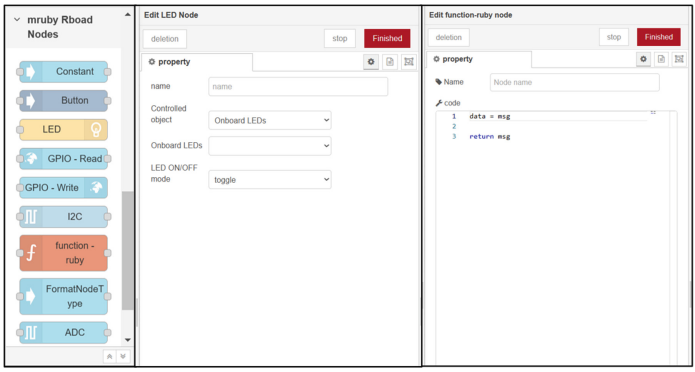


Fig. 5. List of Original Nodes (Left), Edit Node Dialog (Middle, Right)

3.3 Steps from Floe Creation to Execution

The JSON code extracted from Node-RED is converted to byte-code, the byte-code is transferred to the microcontroller using Web Serial API [6], and the program is executed. The following shows the details of the procedure from flow creation to execution in Node-RED. In addition, the screen of Web Serial API is shown in Fig. 6.

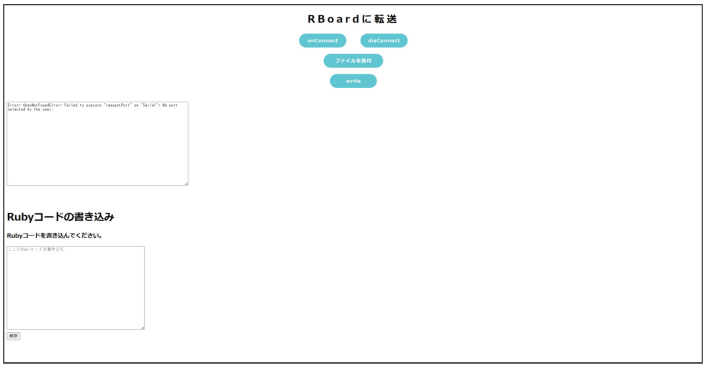


Fig. 6. Transferring Byte-code using Web Serial API

4 Operational Acceptance Testing

Using an ultra-compact analog temperature sensor (MCP9701E), create a program to control LEDs with the temperature sensor and display temperature values on a terminal. A list of the equipment used is shown in Table 2.

Table 2. Equipment Used

Equipment	quantity used
Temperature Sensor(MCP9701E)	1
RBoard	1
MicroUSB	1
LED (Red, Green)	2
Resistance 300[ohm]	2

RBoard is a microcontroller board developed by Shimane Information Processing Center Co. In addition, since it is equipped with a GROVE port, sensors and actuators can be used without soldering [7]. The RBoard was adopted for the operation verification because, as mentioned above, it is easy to handle for first-time users and inexpensive compared to other microcontroller boards.

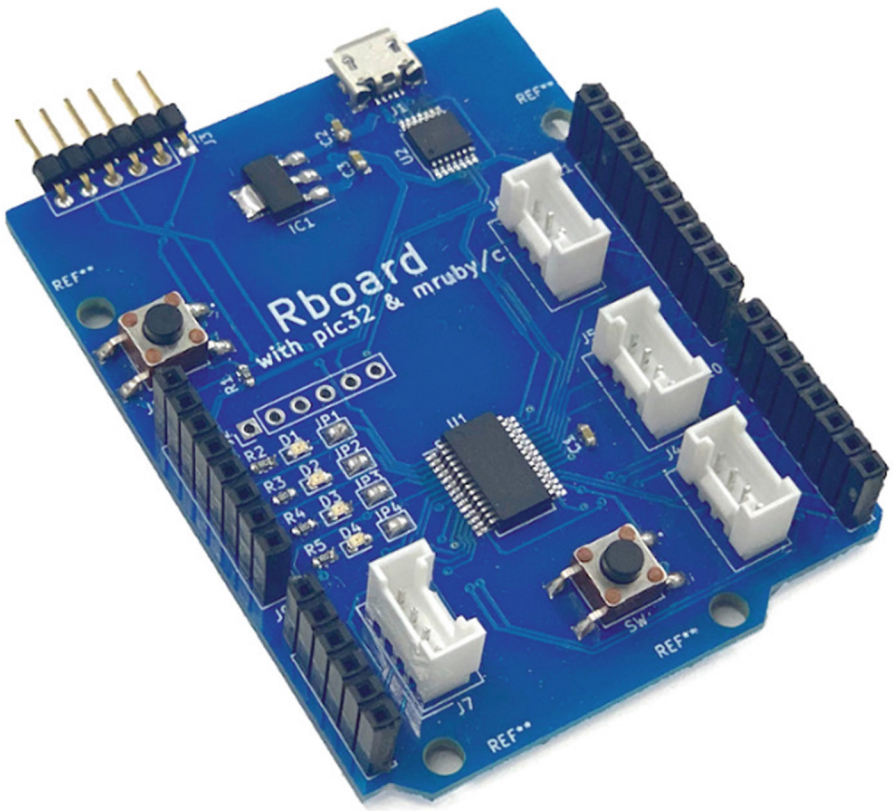


Fig. 7. RBoard

The program created in Node-RED is shown in Code.1.3. The inject node sends data to the right node every second. The GPIO-Read node receives a signal from temperature sensor. Since received signal value from the temperature sensor is an analog voltage, the function-ruby node writes a program to convert it to a temperature in degrees Celsius.

In addition, to confirm that the device behaviour, message is logged by program (Ruby method `puts`).

Of course, We can convert analog voltage to temperature in degrees Celsius using Node-RED nodes instead of using Ruby code, however, the Ruby code is more brief and clear.

Code 1.3. function-ruby-node Code

```
data = msg
temp = data
V2mV = 1000
Vout = 400
temp_coef = 19.5

celsius = (temp * V2mV - Vout)/temp_coef

puts "-----------------------------"
puts celsius
return celsius
```

The temperature data converted by the function-ruby node is sent to the switch node for judgment. In this operation verification, the red LED is turned on when the temperature is 22 °C or higher, and the green LED is turned on when the temperature is less than 22 °C.

Below are the steps for deploying from data-flow description in Node-RED to target device.

1. Place and program nodes on Node-RED to create a flow.
2. Extract JSON code from the created flow.
3. The extracted JSON code is converted to byte-code using the mruby/c compiler. The converted bytecode is downloaded as an mrb file.
4. Click the “onConnect” button on the Web Serial API screen to perform serial communication between the microcontroller board and the PC.
5. Click the “Attach File” button and select the mrb file saved in step 3.
6. Click the “write” button to transfer the binary code to the microcontroller.

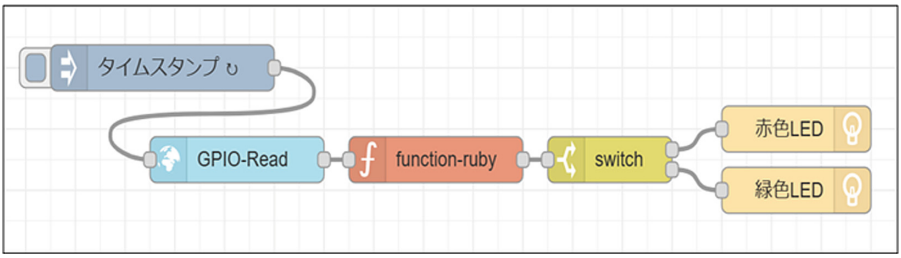


Fig. 8. Program Created in Node-RED

The results obtained from the operational verification conducted are shown below. The temperature values displayed on the terminal and the results of the red and green LEDs lighting up are shown in Fig. 9.

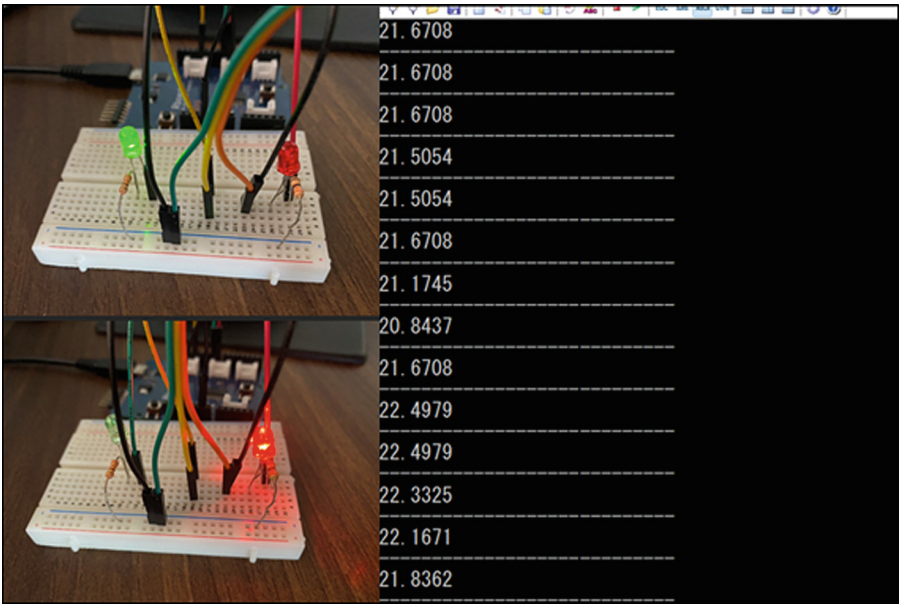


Fig. 9. Temperature below 22 °C (Upper Left), Temperature above 22 °C (Lower Left)

5 Conclusions

Our proposal for embedded software development involves using a data-flow programming environment. The developer creates a flow, which is then converted from JSON code to byte-code using the Ruby code generator mruby/c compiler, with the JSON code serving as data-flow storage information. The converted

bytecode is transferred to the microcontroller, and the program is executed, resulting in an IoT development environment.

We discovered that data-flow programming allows for development irrespective of the programming language, and the created programs are easy to comprehend. As a result, even individuals with no programming experience can explain the processing of embedded systems.

In the future, we aim to implement the node under development and broaden the application's scope to other microcontroller boards beyond RBoard. Furthermore, we intend to test the system's functionality with a robotic arm for industrial purposes using this development environment.

References

1. Tanaka, K., Tsujino, C., Maeda, H.: IoT software by dataflow programming in Mruby programming environment. In: Proceedings of Computational Science and Its Applications ICCSA 2020, Cagliari, Italy, July 1–4, 2020,, Part IV, pp. 212–220 (2020) https://doi.org/10.1007/978-3-030-58811-3_15
2. Tsujino, T., Tanaka, K.: IoT application development for small devices using data flow. In: The 81st National Convention of Information Processing Society of Japan (2019)
3. Murakami, A., Tanaka, K.: Creation of a data-flow programming environment for small devices. In: The 81st National Convention of Information Processing Society of Japan (2022)
4. mruby/c, Shimane Software R&D Center. https://www.s-itoc.jp/activity/research/mrubyc/
5. Node-RED. https://nodered.jp/
6. Web Serial API, W3C Community Group Draft Report. https://wicg.github.io/serial/
7. RBoard, himane Information Processing Center Co. https://www.sjcinc.co.jp/service/rboard 2022