Poster Abstract: Ayaligo: A Programming Framework for Fast IoT System Integration

Pengfei Wang and Zhiwei Zhao
University of Electronic Science and Technology of China
Chengdu, China
{pengfei,zhiwei}@mobinets.org

ABSTRACT

Developing a holistic IoT application that integrates multiple IoT devices is not an easy task. Developers not only need to program the server and multiple embedded boards, but also need to connect them using designated data formats (e.g., JSON) and communication protocols (e.g., MQTT). Besides, programming frameworks are also different for different hardware (e.g., Arduino framework for Arduino UNO and Python scripts for Raspberry Pi), which also increases the development workload. In this regard, this paper presents Ayaligo, a system integration and code generation tool for IoT systems, which allows developers to directly program IoT applications as a whole, using the same syntax for framework-independent and protocol-independent programming. The tool then automatically generate codes that fit different frameworks and protocols by simply modifying the corresponding configurations, and thus can greatly boost the development life-cycle of IoT applications.

KEYWORDS

System integration, IoT, Programming framework

1 INTRODUCTION

A typical Internet of Things (IoT) system usually consists of a series of devices including sensors, actuators and a server for data collection and system commands. Developers need to understand related developing environments, including embedded hardware, embedded firmware, server software, communication protocols, etc. [6]. Such a paradigm usually requires different modules to be implemented by multiple developers with different backgrounds, which apparently introduces inevitable collaboration and synchronization overheads, and further leads to prolonged development lifecycle.

There has been existing efforts trying to address programming differences at the embedded level, such as Arduino [1], MicroPython and CircuitPython [5]. They support hardware abstraction layers (HAL) and provide a unified programming interface for different hardware platforms. Some work such as Cyanobyte in [2] can export different framework code through standardized datasheets. There are also some efforts to program IoT systems as a whole. CodeFirst in [3] can automatically generate hardware configurations based on a node's code. TinyLink 2.0 in [4] can unify programming servers and nodes, but needs to be used with specified cloud servers and clients.

In this paper, we present Ayaligo, a highly abstract system programming framework for IoT integration across heterogeneous hardware, software and communication protocols. In Ayaligo, peripherals, nodes,and servers in a system are abstracted as Python classes, which is intuitive to developers with experience in object-oriented programming. 1) Peripheral classes encapsulate methods for reading and writing data and decorators for event triggering. 2) The node and server classes contains methods for configuring the communication protocols, programming framework, as well as peripheral bindings. When peripherals are bound to both nodes and servers, these classes can be used to automatically establish connections between peripherals and nodes/servers. Based on the above classes and process, Ayaligo can then generate a holistic solution for all IoT devcies, i.e., projects containing the codes and binaries for all involved nodes and servers in the target application. In this way, the development of integrated IoT applications can be greatly simplified.

2 OVERVIEW

Ayaligo framework consists of three parts: Core, CLI, and libraries. The core provides the basic support for programming, the CLI provides the runtime management, and the libraries provide the implementation for project generation.

Ayaligo is based on Python for programming integrated IoT applications, which generates projects for all devices in the application through contextual analysis of the high-level Python codes. Each device's project contains the source code and build configurations or documentations.

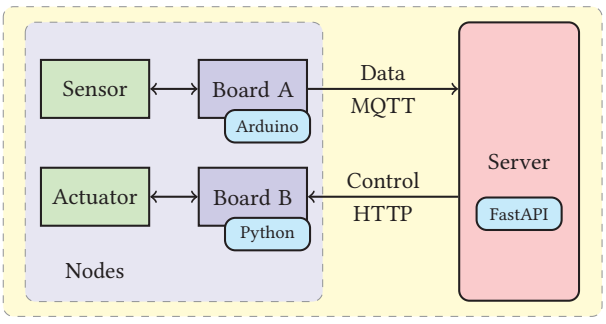

Figure 1: Ayaligo Programming Structure

Figure 1 shows the structure of a typical IoT application programmed and generated by Ayaligo, where the server reads data from the sensor on Board A and then controls the actuator on Board B. Programming languages used Boards A and B are C++ (Arduino) and Python, respectively. Traditionally, a developer needs to write both Arduino C++ codes and Python codes, and combine them with the FastAPI on the server. Now with Ayaligo, the server, boards, and peripherals are defined as classes with their configurations,

979-8-3503-6201-5/24/$31.00 ©2024 IEEE
DOI 10.1109/IPSN61024.2024.00038

indicating the used platforms, languages, and frameworks. Connections are then automatically established based on the board's configuration and bindings.

2.1 Application Development with Ayaligo

Ayaligo uses single code file to program the entire system. The components and boards are instantiated and configured in the code. The server's logic for handling data and events is also written in it. An example of code programmed using Ayaligo prototype is shown in Listing 1.

```
# board and server initialization
board = AtmelAVRBoards.UNO("uno")
server = Server("server")
# components initialization
led = LED("led")
button = Button("button")
# attach components to board and server
board.attach(led)
server.attach(button)
# event handling
@button.on_pressed()
def button_pressed():
    led.toggle()
```

Listing 1: Ayaligo Code Example

Packages for Ayaligo parts will use full type hints. Editor can point out name errors, type errors, and other problems with code hints support. This can also avoid some incorrect usage while editing. With the above codes, an App that control a LED on the board via a button on the server can be developed with Ayaligo, and the projects for both the board and server can be generated according to their specific configurations.

2.2 Project Generation

Ayaligo generates the target projects by context parsing the code file. A context manager will be created before the project generation begins. First, Ayaligo gets the configuration of the boards (server and nodes included), and components from the code, as well as the bindings between them. Then, Ayaligo will generate the project files for each board based on the configuration.

When performing project generation for a board, the specific steps are as follows: 1) get the configurations of components connected to the board, 2) perform resource checking, pin assignments, and document generation, and 4) complete the rendering of the output project based on board's programming framework.

2.3 Extensibility

Ayaligo is designed for framework- and communication-protocol-independent system programming, meaning that it should be very scalable to the add/removal of a new programming framework or a new board is easy to implement. To this end, Ayaligo incorporates the following extensible modules for covering IoT heterogeneity.

Programming framework. For embedded boards, *programming frameworks* can be used by development boards like Arduino and CircuitPython. For the server, *programming frameworks* can be used to develop web or native applications, like FastAPI and Flask.

Communication protocol. *Communication protocols* are essential for connecting IoT boards and servers. Typical protocols are supported as generalized "send" and "receive" modules in Aliyago, such as MQTT, HTTP, and Serial.

Platform and board. Embedded boards and servers are collectively referred to as *platform*. Each embedded *platform* can contain multiple *boards*. For example, the ESP32 *platform* includes the ESP32 Dev Module, ESP32-S3-Box, and other development boards.

Component. Peripherals of embedded microcontrollers are abstracted into *components*. *Components* can correspond to not only electronic components or modules, but also data or functions. For example, a temperature sensor *component* is a hardware module on the embedded board, while temperature data on the server.

Interface. *Interface* describes how the *component* and *board* communicate. For example, serial communication interfaces like IIC, SPI. *Interface* code generation needs to be implemented in *programming frameworks* and will be used directly by *components*.

The above extensible modules are all implementations of the abstract base classes in Ayaligo core, and can be extended by implementing the abstract methods. Each *board* or *component* class has the same interface, and unified code generation is achieved through different parsing configurations inside the library. Besides, *components* can additionally implement unique methods or decorators to make them more intuitive.

3 EXPERIMENTS

We implemented Ayaligo prototype, and invited four volunteers (two amateur and two experienced) to develop two IoT applications, one of which uploads temperature data from a node to the server, and the other displays temperature data from two nodes on a third node's screen, forwarded through the server. Both use Arduino framework and FastAPI framework. The results show that: 1) Ayaligo typically reduces the number of pure code lines of by 70% and development time by 75% for developers. The reason is that Ayaligo avoids partial duplication, especially structuring and initialization code in Arduino framework; 2) Ayaligo can be used in complex system programming by using Python code, which also makes system applications easier to maintain.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (No. 62372094).

REFERENCES

[1] Steven F Barrett. 2022. *Arduino III: Internet of Things.* Springer, Cham, Switzerland.
[2] Nicholas Felker. 2020. Design of Cyanobyte: An Intermediate Representation to Standardize Digital Peripheral Datasheets for Automatic Code Generation. In *2020 IEEE Sensors Applications Symposium (SAS).* IEEE, Kuala Lumpur, Malaysia, 1–6.
[3] Daniel Graham and Gang Zhou. 2016. Prototyping wearables: A code-first approach to the design of embedded systems. *IEEE Internet of Things Journal* 3, 5 (2016), 806–815.
[4] Gaoyang Guan, Borui Li, Yi Gao, Yuxuan Zhang, Jiajun Bu, and Wei Dong. 2020. TinyLink 2.0: integrating device, cloud, and client development for IoT applications. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking (MobiCom '20).* ACM, New York, NY, USA, Article 13, 13 pages.
[5] MicroPython & CircuitPython contributors. 2024. CircuitPython. https://docs.circuitpython.org/
[6] Usman Raza, Parag Kulkarni, and Mahesh Sooriyabandara. 2017. Low Power Wide Area Networks: An Overview. *IEEE Communications Surveys & Tutorials* 19, 2 (2017), 855–873.