PrIoT Demo: Example of Invariant Functionalities

Nahit Pawar
Institut Polytechnique de Paris
Télécom SudParis
nahit.pawar@telecom-sudparis.eu

Thomas Bourgeau
KB DIGITAL AG
Switzerland
tbourgeau@kb-digital.ch

Hakima Chaouchi
Institut Polytechnique de Paris
Télécom SudParis
hakima.chaouchi@telecom-sudparis.eu

***Abstract*—This paper presents a simple IoT scenario that uses the concept of application invariant functionalities. We propose to implement our scenario with the PrIoT framework that provides a lightweight approach when designing intermediate abstraction layers with minimalist programming functionalities that are IoT application invariant.**

***Index Terms*—IoT system architecture, device heterogeneity, invariant functionalities**

I. Introduction

In the context of the Internet of Things (IoT), device and communication protocol heterogeneity poses serious problems for interoperability control and also for harmonized IoT development tools over a plethora of hardware from various manufacturers. As exposed in our current studies [4], irrespective of IoT domain and its requirements an IoT end-system performs some basic high-level invariant functionalities as shown in Table I . In this paper, we propose to implement a simple IoT scenario using the invariant functionalities exposed by our PrIoT framework [3] and is explained hereafter.

TABLE I: PrIoT application invariant functionalities

Application file (*.app) - PrIoT-Lang	
Read	Read data from sensor
Write	Write data to actuator
Execute	Execute user defined operations on data
Send	Send data using transceiver
Receive	Receive data from transceiver
Wait	Wait for defined period

II. PrIoT Framework Overview

The PrIoT framework aims at leveraging IoT adoption and usage from design to deployment and better handle the heterogeneity property of the IoT device, services and applications. The main design philosophy behind our PrIoT framework is "code once port anywhere". In this section we focus only on the invariant functionalities exposed by the PrIoT language and its configuration features. Further implementation details, documentation and example scenario can be found in its community website [1].

In PrIoT the application logic is independent of any IoT end-system hardware, this implies that the same application logic can be ported to any hardware supported by PrIoT.
The IoT application developer implements the application logic in application files **(*.app)** and hardware or software libraries configuration as per application requirements in configuration files **(*.cfig)** as it will be shown in the use case below (Section III). The application file and configuration file functions are represented in Table I and Table II respectively.

The configuration file includes - selecting IoT end-devices, standard communication protocol and interface between IoT end-devices. These configuration files have all the necessary information required by PrIoT to automatically configure and compile the whole IoT application on selected embedded system and hardware.

PrIoT is equipped with high level programming language - PrIoT-Lang, which exposes user with device independent high level set of functions that are kept limited but capture most practical IoT scenarios. The high level language uses a procedural c-like structure with conditional operators and inherit from the wiring language syntax [2].

The first step before implementing the application logic is to select hardware components from PrIoT Database (PrIoT-DB), this step does not require to know the exact specification of hardware vendor but rather it defines what are the high level hardware entities used by application logic.

TABLE II: PrIoT Configuration functionalities

Configuration file (*.cfig) - PrIoT-Config	
Select	Select hardware components
Import	Import hardware and software library
Define	Define hardware interface, software variable and library configuration.

III. Scenario Implementation

In order to show the added value of using invariant functionalities with our PrIoT framework, in this section we explain with an IoT scenario usecase what are the components to be developed and the implementation workflow used.

A. Proposed Scenario

We present in Fig. 1 our simple IoT scenario that consists of a device equipped with a temperature sensor that sends its measured data periodically to an MQTT broker that is hosted on a gateway. A wireless communication channel using 802.11 WiFi is used where the gateway acts as an access point and the device is configured as a station. In the spirit of coding tutorials, this example can be seen as our *Hello World* scenario to showcase the core concepts of the PrIoT framework and its relation to invariant functionalities. For sake of simplicity, we will only focus on how to implement the connected device with PrIoT but the gateway and other more evolved scenario can be implemented with PrIoT by following the same workflow.

978-3-903176-32-4

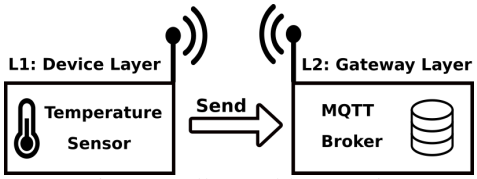

Fig. 1: Hello world Scenario

B. Demo Requirements

As stated before, PrIoT provides a large set of target hardware and MCUs and allows to easily change underlying hardware targets without modifying the application logic. In order to use the PrIoT framework, it is mandatory to download it's command line interface as described in its website. For our scenario, the connected device is made of an Arduino Uno board based on the ATmega328P, an Espressif ESP8266 as wireless transceiver and a Negative Temperature Coefficient (NTC) thermistor to sense the temperature. For the Gateway, a Raspberry Pi with a Mosquitto MQTT service is set and configured as a WiFi access point.

C. Scenario implementation with PrIoT

```
# File name - hello_world.cfig

Select Sensor.Temperature as temp
Select Transceiver.IEEE80211 as wifi
Select Hardware.MCU as avr

# Communication Protocol
Import Communication.80211 as wc
Import Gateway.PubSub.client as mqtt

# Embeded System
Define avr.Type as ATmega328p
Define avr.GPIO.RX4 as wifi.TX
Define avr.GPIO.TX5 as wifi.RX
Define wifi.SSID as "My_SSID"
Define wifi.Mode as "Station"
Define mqtt.Connector as wifi
Define mqtt.Topic as "MyRoom/Temp"
Define mqtt.Host as "mqtt.myhome.local"
Define timer as "2s"
```

Listing 1: Configuration file

In our example scenario, the configuration file and the application file for the object is shown in Listing 1 and 2 respectively.

In the configuration file, the **Select** keyword allows user to select hardware from the repository of IoT-end devices maintained by PrIoT-DB, this also includes any relevant devices libraries used by the application. The **Import** keyword is used to inform PrIoT which library to include in the project, in this example we included standard communication library for IEEE 802.11 and PubSub. At this stage, the selected elements are defined as default element from the PrIoT-DB and are not linked to a specific hardware vendor. Thus for the object, a generic temperature sensor, a WiFi transcriever have been selected as electronic components and communication protocols such as a Wifi Station mode and PubSub Client have been selected as communication libraries.

We also have the ability to specify dedicated hardware and configuration at this stage by using the **Define** keyword allowing user to define precise hardware for IoT-end devices. In our case, the targeted MCU and its connection to the generic transceiver are defined in addition to the wireless communication configuration and the PubSub setup. We can also define variable to be used in the application file such as the *timer* variable exposed in our scenario configuration file.

After having defined the IoT components to be used, we can program the targeted scenario through the application file that consist of application logic written using device independent language - PrIoT-Lang. To be coherent with the hardware components selected, we have to use the **Import** keyword and specify the related configuration file to be imported. In Listing 2 we see that the object sense the temperature and sends to the MQTT broker the temperature information published on the given topic and wait during a certain time defined by the timer variable before repeating the same actions.

```
# File name - hello_world.app
Import hello_world.cfig

T = temp.Read()
mqtt.Send(T)
Wait(timer)
```

Listing 2: Application file

Following this approach, we have the ability to program a scenario with default hardware specification. As not all hardware are similar, it is possible to target a more precise hardware components configured with the **Define** keyword.

In order to generate the binary code from the application and configuration files we will use the PrIoT command line as shown in Listing 3.

```
priot parse hello_world
priot build hello_world
```

Listing 3: PrIoT command line

The first line use the parse argument with the application file name to produce a single file with linked libraries from PrIoT-DB. Then, the second line use the build argument to generate a binary file for the target device MCU.

IV. Concluding Remarks

In this paper, we have presented our PrIoT framework invariant functionalities and demonstrated it's usage through a simple IoT scenario. With the same logic and simplicity, it is possible to implement more complicated IoT scenario spanning a large set of IoT application with their invariant functionalities.

References

[1] Priot website. http://www.priot.org. [Online; last accessed 25-Jan-2021].
[2] H. Barragán. Wiring: Prototyping physical interaction design. *Interaction Design Institute, Ivrea, Italy*, 2004.
[3] N. Pawar, T. Bourgeau, and H. Chaouchi. PrIoT: Prototyping the Internet of Things. In *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 216–223. IEEE, 2018.
[4] N. Pawar, T. Bourgeau, and H. Chaouchi. Study of iot architecture and application invariant functionalities. In *IEEE/IFIP International Symposium on Integrated Network Management*. IEEE, 2021.