Exploring Bluetooth Communication Protocols in Internet-of-Things Software Development

Tri Minh Triet Pham, Jinqiu Yang
Computer Science and Software Engineering
Concordia University, Montreal, Canada
{p_triet, jinqiuy}@encs.concordia.ca

***Abstract*—Internet of Things (IoT) development heavily depends on the connectivity of real-world objects. Bluetooth technology is widely applied to such connectivity in many IoT domains, such as smart home systems. Developing an integrated IoT system involves various stakeholders, e.g., mobile app developers and firmware developers. Discrepancies on the connectivity of the devices, i.e., how to communicate, may occur between different stakeholders in IoT development. Discrepancies occur when one group of developers misunderstand the communication protocols or incorrectly implemented them in the code. Such discrepancies may lead to unmet requirements and runtime connection errors. To help reduce such discrepancies, we perform a study to understand the current practices of designing *Bluetooth communication protocols* (BCPs) (i.e., by firmware developers) and how software developers manage the diverse BCPs in the code. Such understanding is a first step to provide tool support that can help developers better manage BCPs and detect (fault-indicating) discrepancies, aiding the maintenance effort of mobile applications.**

***Index Terms*—Internet of Things, Bluetooth, mobile applications, software maintenance**

I. Introduction

Internet of Things (IoT) is an emerging technology that aims to connect physical objects from diverse domains. An example of an IoT application is the concept of a smart home, where devices such as light fixtures, thermostats, and other smart appliances are connected to make intelligent decisions. Among the various wireless technologies adopted by IoT [1], Bluetooth low energy (*BLE*) is an integral part of the IoT paradigm as it is one of the main communication channels between smartphones and embedded devices. An example of such connectivity is between a smart device (e.g., a fitness tracker) and a smartphone through an application that is programmed to communicate between the two devices. Hence there has been an increasing need for developing software components that rely on BLE for communicating with embedded devices, according to 2019 Bluetooth market update [2].

Enabling BLE connectivity in software requires compliance with the *Bluetooth communication protocols* (BCPs) defined for an embedded device. BCPs specify how one software (e.g., a mobile application) can communicate with an embedded device. Typically, firmware developers define and program BCPs in embedded devices, and software developers (e.g., mobile application developers) leverage the defined BCPs when developing software to communicate with embedded devices. For example, the BCPs of the Mi scale specify that upon receiving the stop command (*0x03*) from the scale, the mobile application should start a cleanup procedure. Correspondingly, to comply with the BCPs, the mobile application should send (write) a byte value (*0x03*) to the corresponding characteristic (i.e., represented by *Java.util.UUID*) on the Mi scale to acknowledge the receipt of the stop command (as shown in line 2). In this work, we call such a pair of characteristic UUID and value that performs a specific task one *command*, which is the basic unit in BCPs. Note that the value can be a dynamic value such as date or temperature. Implementing a functionality may require a sequence of commands, even combined with conditions.

```
1 if (data[0] == 0x03) {//then clean up procedure is
      required
2   writeBytes(WEIGHT_MEASUREMENT_SERVICE,
        WEIGHT_MEASUREMENT_HISTORY_CHARACTERISTIC, new
        byte[]{0x03});
3 ...}
```

Failing to follow BCPs can result in silent failures in the software and creates difficulties in debugging due to the less systematic nature. BCPs are the interfaces to interact with embedded devices; however, compared to library APIs, when violations with BCPs occur, there may not be explicit compilation errors or runtime failures that are observable to developers.

The prevalence and likelihood of BCP violations vary depending on the complexity of a BCP (i.e., how firmware developers define the BCPs of one device) and the software's programming practices (i.e., how software developers manage compliance with BCPs). To further reveal the challenges in utilizing Bluetooth in IoT development, we take the first step to study BCPs. We answer the following two questions.

- **RQ1**: What are the common styles to define Bluetooth communication protocols (BCPs) by *firmware developers*?
  In this RQ, we explore the common structures of BCPs. Firmware developers have much freedom to design BCPs. The design of BCPs may vary even for similar devices.
- **RQ2**: How are Bluetooth communication protocols (BCPs) managed in software by *software developers*?
  For different BCP styles, software developers may utilize different programming practices to facilitate compliance with BCPs, i.e., making the implementation more systematic and easy to maintain.

TABLE I: Summary of the two open-source mobile applications in F-Droid that are suitable for this study.

Mobile Applications	Type of Embedded Devices	# of Supported Devices
openScale	Weight Scale	20
Gadgetbridge	Smartwatch	15

II. Experiment Design

Studied subjects. To answer the RQs, we need two types of data: BCPs in written format and the corresponding application Bluetooth components. We started collecting data from FDroid since it is one of the largest collections of open-source Android applications. Although many applications in FDroid use Bluetooth, many do not make BCPs available. Alternatively, we can reverse engineer BCPs from the applications, but we decided to leave that for future work. Hence, in this work, we identify two applications with written BCPs (i.e., in the documentation) in total. Both applications work with multiple devices: 20 different scales in openScale and 15 different smartwatches in Gadgetbridge (Table I).

Manual analysis. We examined the documentation and the source code that implements the BCPs in openScale and GadgetBridge to understand how developers manage the compliance with BCPs. The documentation describes BCPs in detail; however, it lacks proper structure (i.e., almost in free-format). For each device, we read all the developers' documents concerning Bluetooth communication and extracted the BCPs. After we have analyzed the provided documentation for 35 devices as shown in Table I, we found the BCPs for seven devices from the two studied projects. For each document, we located all the mentioned BCPs and recorded its details, including the type of action (send or receive), the characteristic UUID, the constraints of the value, the relevant functionality, and the response value (if any).

III. Research Question Results

Results of RQ1. We concluded three common styles that firmware developers use to design BCPs. *Style A*: All of the commands (of a device) use one characteristic, and the first few bytes of the value is the unique identifier to communicate with the device stating the purpose of the command. *Style B*: All of the commands use one of two characteristics: one for receiving and one for sending. Similar to Style A, for the commands under the same characteristic, the first few bytes of the value specify the purpose of the command. *Style C*: Each command uses a unique characteristic. The value expresses command-specific information during the interaction between the software and the device. Table II shows the number of commands of each studied device and its BCP style. All three styles are ad-hoc to some extent. Style C heavily relies on the uniqueness of characteristic UUID. However, characteristic strings are long and meaningless, thus difficult to maintain. For example, a characteristic in Mi scale is *00002a2f-0000-3512-2118-0009af100700*. Differently, Style A simplifies the use of characteristic; instead, it heavily uses the first few bytes of the value, which are shorter than a characteristic string, but still meaningless, and often requires bit-wise substitution. Style B is a balance between Style A and C.

TABLE II: Summary of the commands and BCP style identified based on documentation.

Device	Type	# Commands	BCP Style
Beurer	Weight Scale	47	A
Sanitas	Weight Scale	47	A
Trisa	Weight Scale	8	B
HPlus	Smartwatch	8	B
Xiaomi	Weight Scale	9	C
Medisana	Weight Scale	3	C
ZeTime	Smartwatch	15	C

Results of RQ2. Motivated by the uncovered styles in RQ1, we continued to analyze how software developers manage the compliance of BCPs. For all styles, the common practice is to assign the characteristics to class fields that are final. The identifiers of such class fields are very important for developers in both implementation and maintenance. Style A and B need extra class fields to store the unique identifiers used in each command. Furthermore, the first few bits of a value may be substituted before communicating with the device. For example, when communicating with a Beurer scale, the first four bits of the byte value is substituted with an identifier, e.g., *binary & 0xF0 | unique_id*.

IV. Related Work

There has been some work in discussing wireless and network technologies in IoT development [1]. However, we find only a few studies on the software development part of IoT development. A recent work by Aly et al. [3] studied the lack of interoperability among various technologies adopted by IoT development. Their study reveals that interoperability may lead to challenging integration problems in software development in IoT systems.

V. Conclusion

We present a first study to understand the design and management of Bluetooth communication protocols in IoT software development. Our study uncovers common styles of designing BCPs, all of which do not provide sufficient resilience on possible human errors. In addition, we study how software developers manage to comply with BCPs by analyzing connectivity-related code.

References

[1] Z. Sheng, S. Yang, Y. Yu, A. V. Vasilakos, J. A. Mccann, and K. K. Leung, "A survey on the ietf protocol suite for the internet of things: standards, challenges, and opportunities," *IEEE Wireless Communications*, vol. 20, no. 6, pp. 91–98, December 2013.

[2] "Bluetooth market update." [Online]. Available: https://3pl46c46ctx02p7rzdsvsg21-wpengine.netdna-ssl.com/wp-content/uploads/2018/04/2019-Bluetooth-Market-Update.pdf

[3] M. Aly, F. Khomh, Y. Guéhéneuc, H. Washizaki, and S. Yacout, "Is fragmentation a threat to the success of the internet of things?" *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 472–487, Feb 2019.