Demo: Integrated Development of IoT Applications with OneLink

Gaoyang Guan, Yuxuan Zhang, Borui Li, Wei Dong, Yi Gao, Jiajun Bu
College of Computer Science, Zhejiang University, China, and,
Alibaba-Zhejiang University Joint Institute of Frontier Technologies
{guangy, zhangyx, libr, dongw, gaoy}@emnets.org, bujj@zju.edu.cn

Abstract

The recent years have witnessed the rapid growth of IoT (Internet of Things) applications. A typical IoT application usually consists of three essential elements: the device side, the cloud side and the client side. The development of a complete IoT application is very difficult because it involves drastically different technologies and complex interactions between different sides. In this demo, we present OneLink, which integrates device, cloud and client side IoT development in a single piece of code. OneLink provides a coherent C-like programming language for developers and a programming model that separates the application logic and the underlying services. It also extends existing IFTTT syntax by adding new keywords and data structures in order to express diverse IoT applications. Moreover, OneLink can automatically tailor TinyApps and policies to the application needs in order to optimize the performance by using both static and dynamic optimizers. We illustrate how to write a piece of IoT application code with OneLink by going through developing smart home applications.

1 Introduction

The recent years have witnessed the rapid growth of IoT applications, including environmental monitoring [4], shared bicycles [3], and smart homes [7]. A typical IoT application is usually composed of three fundamental elements: (1) The device side: the IoT device perceives its surrounding environment and sends the sensing data to the gateway which can be a WiFi access point, a cellular base station, etc. (2) The cloud side: the cloud provides key functionalities, including data storage, device management, computation-intensive tasks calculation, etc. (3) The client side: the mobile device which interacts with the end users.

Developing such an application is thus very difficult because developers need to deal with drastically different technologies and the complex interactions between the different sides. TI Inc. has pointed out that "*IoT application development needs to be easy for all developers, not just to experts*" [1]. How should we simplify the IoT development process especially for non-expert developers?

The above challenge has attracted significant attention from both academia and industry. The academia usually focuses on improving one particular aspect of the system. For example, TinyLink [5] provides a top-down approach which greatly accelerates the device side IoT development. Beam [7] provides middleware service to simplify sensor inference, without detailed consideration on the implementation on the device and client side. The industry (e.g., IBM Watson IoT) usually focuses on providing an IoT development platform which ties different aspects together while leaving the particular implementation details.

Is it possible to implement a complete IoT application using one piece of code expressing the holistic logics on device, cloud and client sides in a coherent way? If that is possible, a single application developer can quickly develop a complete IoT application.

In this demo, we present OneLink, which integrates device, cloud, and client side development in a single piece of code. OneLink builds on top of many existing works. For example, OneLink uses IFTTT-like rules [6] to specify IoT application logic; OneLink builds on top of TinyLink to generate the hardware configuration as well the actual binary program for the corresponding hardware. The existing works shed light on many important aspects towards the goal of simplifying the IoT development.

2 OneLink Overview

In order to design a single coherent language for developing a complete IoT application, we intend to provide the following features for the C-like OneLink programming language: (1) A separation between the application logic and the underlying services. The key abstractions OneLink provides are IFTTT-like rule capsules called policies and service capsules called TinyApps; (2) A simple and clear method for specifying the internal interactions, e.g., data flows and service calls; (3) Easy-to-use programming APIs for developing the application and designing the UI.

Moreover, OneLink incorporates a dynamic optimizer and a static optimizer which aim to optimize the application performance at compile-time and runtime, respectively. We

International Conference on Embedded Wireless Systems and Networks (EWSN) 2019
25–27 February, Beijing, China

ISBN: 978-0-9949886-3-8

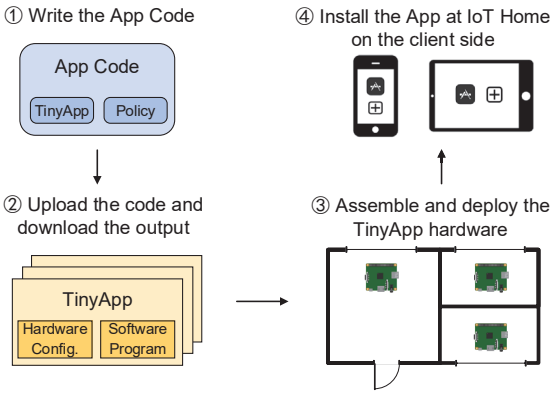

Figure 1. OneLink user workflow.

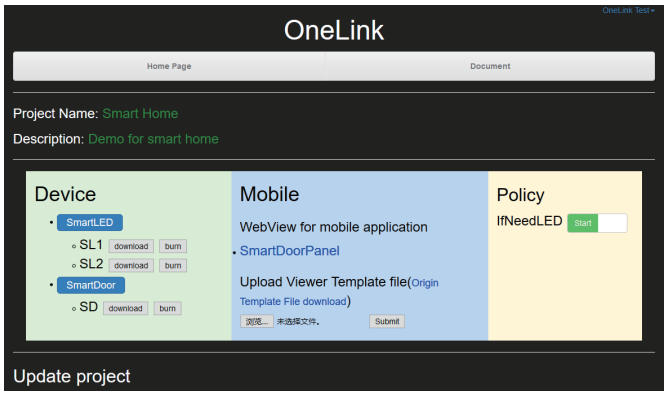

Figure 2. OneLink website.

believe that it is reasonable to tailor TinyApps and policies on different sides to the application needs in order to optimize the performance (e.g., energy consumption). For example, OneLink dynamic optimizer on the cloud analyzes the trigger conditions of IFTTT-based policies and dynamically acquires sensor data with conditional-aware intervals in order to reduce energy consumption. Static optimizer transforms the rules on the cloud side to the TinyApp code on the device side at compile time in order to suppress frequent data acquisitions and reduce the energy consumption.

3 Demo Description

In this demo, we illustrate how to write a piece of IoT application code with OneLink by going through developing a smart home application. The audience can also create their own IoT applications at the scene. The sample source code can be downloaded from the site [2]. The audience can observe these results from the application: (1) Automatic generation of applications at the IoT, the client and the mobile sides from one piece of OneLink code; (2) Visualization of sensing data (e.g., ambient light) and triggered events (e.g., opening the door and home occupancy) from different TinyApps on the generated mobile application; (3) Interactions, e.g., turning LEDs on/off according to door and home occupancy events which are described in IFTTT-based policies, across applications on different sides; (4) Consequences of changing the policies at runtime such as parameters of the IFTTT conditions; (5) Dynamically optimizations, e.g., changes of condition-aware sensor polling intervals at runtime according to the sensor values.

Figure 1 shows the development process which contains the following four steps: (1) **Write the application code**. We will provide the above sample code on our OneLink website whose URL is encoded in a QR code at the scene. The audience can participate from the beginning by scanning the QR code. (2) **Upload the code and download the output**. Figure 2 shows the applications on the three sides, including the hardware configuration (e.g., connection figure), each TinyApp binary program which can be burnt to the TinyApp hardware, the WebView for the mobile client and the IFTTT-based policies. (3) **Assemble and deploy the TinyApp hardware**. We will show the audience how we assemble the hardware platform from COTS hardware components with the help of connection figures. As an alternative, we also provide already assembled hardware platform which are deployed on the stage property, i.e., a tiny plastic box which mimics a small room. (4) **Publish the application and watch the results**. Finally, we will publish the application at the mobile side to the OneLink IoT Home, where the audience can observe the aforementioned results.

4 Conclusion

In this demo, we present OneLink, a novel system which integrates device, cloud and client side IoT development in a single piece of code. OneLink provides a C-like programming language so that developers can express diverse IoT application logics in a coherent way. OneLink extends IFTTT syntax by adding new keywords and optimizes application performance with the static and the dynamic optimizers. Currently, we are working on how to simplify writing IFTTT-based policies. We intend to enable virtual sensors that combine different physical sensors in order to provide new functionalities, e.g., detecting specific human activities.

5 Acknowledgments

This work is supported by the National Science Foundation of China (No. 61872437) and the Fundamental Research Funds for the Central Universities (No. 2018FZA5013).

6 References

[1] Challenges in the Internet of Things. http://www.ti.com/ww/en/internet_of_things/iot-challenges.html.

[2] Example Code. https://www.dropbox.com/sh/v8m2tay5s1ip54o/AABvrPyv8diW3iHKXarb-pW1a?dl=0.

[3] J. Froehlich, J. Neumann, N. Oliver, et al. Sensing and predicting the pulse of the city through shared bicycling. In *Proc. of IJCAI*, 2009.

[4] Y. Gao, W. Dong, K. Guo, X. Liu, Y. Chen, X. Liu, J. Bu, and C. Chen. Mosaic: A low-cost mobile sensing system for urban air quality monitoring. In *Proc. of IEEE INFOCOM*, 2016.

[5] G. Guan, W. Dong, Y. Gao, K. Fu, and Z. Cheng. Tinylink: A holistic system for rapid development of iot applications. In *Proc. of ACM MobiCom*, 2017.

[6] S. Heo, S. Song, J. Kim, and H. Kim. Rt-ifttt: Real-time iot framework with trigger condition-aware flexible polling intervals. In *in Proc. of IEEE Real-Time Systems Symposium (RTSS)*, 2017.

[7] C. Shen, R. P. Singh, A. Phanishayee, A. Kansal, and R. Mahajan. Beam: Ending monolithic applications for connected devices. In *Proc. of USENIX Annual Technical Conference*, 2016.

A Hardware Used

In this demo, we will use one WiFi AP, four mainboards (i.e., three Arduino Mega2560 and one Raspberry Pi 3 model B+), three extension boards (i.e., Arduino Mega Shield), more than twenty peripherals (e.g., sensors, actuators and communication modules), other accessories(e.g., dupont lines, RJ45 Ethernet cables and power strips) and stage properties (e.g., a 0.5m*0.3m*0.2m plastic box). We also use the cloud service hosted by Alibaba Cloud in Hangzhou. In addition, we will use a tablet and a laptop which are used to give better visualization effects. We will bring all these hardware with ourselves in the demo session.

B Space Needed

Our demo only requires a normal table where we can put all these hardware. No additional space is needed.

C Special Equipment

We do not need special equipments besides the hardware mentioned above.