Open Software Platform for Companion IoT Devices

Hyemin Lee, Dongig Sin, Eunsoo Park, Injung Hwang, Gyeonghwan Hong, and Dongkun Shin
Sungkyunkwan University, Suwon, Korea
{gudbooy, dongig, pes9488, sinban, redc7328, dongkun}@skku.edu

***Abstract*-- In recent years, a variety of IoT devices have been introduced in market. However, most of the IoT devices are designed to perform dedicated functions and cannot be programmed by consumers. In this paper, we propose an open IoT platform which enables users to implement various services swiftly and easily via JavaScript API set. In particular, the platform is designed for the companion IoT devices, which can be accessed and controlled by mobile devices such as Android smartphone. The proposed software platform for companion IoT devices equips with the various state-of-the-art techniques in application management system, sensor, camera, and network.**

I. Introduction

Recently, many IoT (Internet-of-Things) services and devices have been emerging. In order to implement IoT services and devices quickly, an efficient and versatile IoT software platform is required. Generally, each IoT device is designed to perform dedicated functions and it is not programmable. Therefore, a new device is required to use new IoT services. However, many new IoT services are introduced every day, and user wants to use multiple IoT services. In particular, user even wants to implement his own applications. Therefore, a programmable IoT software platform is essential for the proliferation of IoT services. Although there are several programmable software platforms for mobile devices such as Android, iOS, and Tizen, they are too heavy for resource-limited IoT devices and it is difficult to customize them for IoT devices. In addition, high programming skills are required to implement applications. In this work, we introduce a novel programmable software platform for IoT devices, called *OPEL (Open Platform Event Logger)*. Especially, since many IoT devices on the market are based on companion model, the OPEL platform supports the mode such that the application on OPEL platform can easily cooperate with mobile devices such as Android-based smartphones.

The OPEL software platform is designed considering the following requirements. First, the IoT platform should be programmable with an easy programming language. In particular, the programming language should have high productivity, portability, and extendibility to enlarge the IoT ecosystem. A good candidate is JavaScript (JS) language. Second, the platform should provide high-level APIs for easy application development. The APIs need to provide several functions including sensor and device management, communication, etc. Third, the platform should support

This research was supported by the MSIP, Korea, under the G-ITRC support program (IITP-2016-R6812-16-0001) supervised by the IITP.

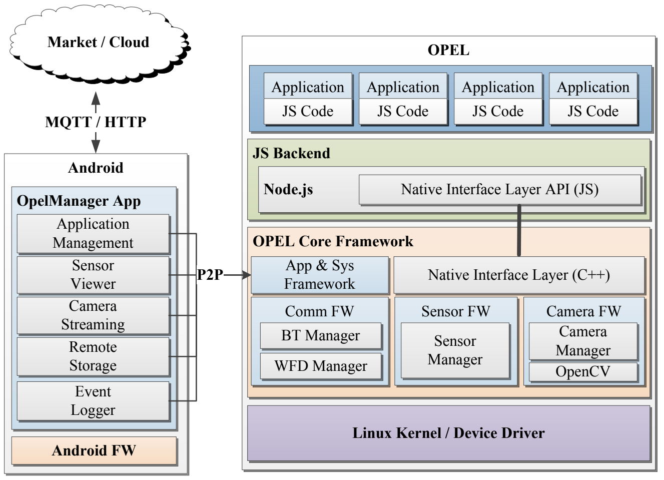

Fig. 1. OPEL Software Architecture

multiple IoT applications. User should be able to install multiple applications with different functions, which can be executed concurrently. Then, user can exploit multiple services on single device. Lastly, the software platform should enable the companion IoT device to communicate with its host device (e.g. Android smartphone), and to be controllable via its host device.

In order to satisfy these requirements, the OPEL platform is designed to provide the following features. First, OPEL includes the JS runtime framework called Node.js as many IoT platforms such as IoT.js [1] and Mongoose IoT [2] have adopted JS language. Second, in order to provide the high-level JS device API, *OPEL core framework* (*OCF)* is provided, which can provide several services to applications. The inter-process communication (IPC) between JS API and OCF is performed by *Native Interface Layer* (*NIL*). Third, for efficient management on multiple JS applications, the life-cycle and package management systems are performed by the App & Sys framework (F/W). Lastly, the OPEL platform also provides an Android application, called *OpelManager,* which is used to control OPEL IoT device.

In addition to the fundamental features, the OPEL platform enables a JS application to access several sensor devices. The *sensor framework* provides the centralized control to concurrent sensor acquisition requests. The Sensor F/W supports sensor plug and play. The *camera framework* supports the control on camera device. In particular, it enables multiple camera applications can run concurrently at a resource-constrained device by deploying a proxy daemon process which manages the sharing of camera frame buffers among concurrent applications [3].

IoT devices usually do not have user-interactive I/O devices such as touch-screen, keyboard, etc. In order to overcome the limitation, the OPEL platform provides the OPEL platform provides the *Remote System Protocol* (RSP) which enables OPEL device to be controlled by Android mobile device via Android UI and system APIs.

The communication framework can support dynamic network protocol selection between Bluetooth (WPAN) and Wi-Fi Direct (WLAN) based on user-context for energy-efficient and high bandwidth communications.

II. Design And Implementation

Fig. 1. describes OPEL software platform architecture. The Node.js is adopted for JS application runtime. OCF is a native service daemon (out-of-process model) implemented in C/C++. In order to support IPC between a JS App and OCF, we exploit the Node.js *Addon* function which provides an interface with native language-based dynamically-linked shared objects. We implemented IPC protocol based on D-Bus in the JS Add-on function. The following subsection describes the detailed design of OCF.

A. Application & System Framework

The App & Sys F/W provides two main functions: appl-ication life-cycle management and communication with Android mobile phone. The life-cycle management includes the handling on installation, launching, and termination of application. During application installation, a manifest file in the application package is parsed by the App & Sys F/W, which contains application information such as unique application ID, JS installation path, and API access authority. The information is recorded at the database of OPEL device. If a JS application is executed, its process ID (PID) is stored in *process management table* which keeps tracking the life-cycle of the application. If user invokes a termination procedure, the F/W sends a signal to the target process. Then, a corresponding JS callback will be invoked. Another function of App & Sys F/W is to support the remote system protocol (RSP). Through the RSP, the UI and system APIs of host mobile phone can be invoked remotely by the JS RSP APIs in OPEL device. The RSP APIs are useful for the IoT products without user-interactive I/O devices. For the remote UI handling, JS programmer simply specifies the name of Android UI objects and the corresponding values as JS function parameters. Then, the App F/W makes UI object tree and send it to host mobile phone, which parses it UI object tree and draws the UI objects dynamically.

B. Sensor Framework

Since sensor is a key component in IoT platforms, OPEL supports a software-level plug-and-play of sensor drivers with an abstraction interface layer for sensor operations. The required sensor drivers are dynamically loaded by model-driven JS API. There are three types of sensor acquisition JS APIs: Synchronous, Asynchronous, and Periodic APIs. The sensor F/W supports sensor request merging. When several applications use different sensor sampling rates, the sensor F/W merges them into one with the highest sampling rate in order to eliminate the duplicated requests and reduce energy consumption.

C. Camera Framework

Most of the modern operating systems (e.g. Linux) allow only one process to exclusively access the camera devices [3]. However, the camera F/W of OPEL supports the concurrent execution of multiple camera applications. For example, video recording, snapshot, vision application, and streaming service can be executed concurrently with one camera device. A proxy daemon for framebuffer allocates the camera frames to shared memory buffer such that multiple applications can share them.

D. Communication Framework

The communication framework transfers the network request between OPEL device and Android mobile phone. Even when OPEL device is not directly connected to stable internet network, the network requests for accessing cloud service can be sent via the mobile phone. When the JS application running at OPEL device calls the cloud APIs, the requests are passed to a cloud server via mobile phone. Currently, the MQTT protocol is used for connection with cloud server. To provide energy-efficiency and high bandwidth communication, the communication F/W selects Bluetooth or Wi-Fi Direct dynamically as communication module considering their bandwidths and energy consumptions. For instance, when the size of a message exceeds the predefined threshold, the communication is switched from Bluetooth to Wi-Fi Direct.

III. Usage Scenarios

Many IoT services can be easily implemented with the OPEL software platform. For instance, we can implement a sensor viewer application with only 50 lines of JS code. A smart gardener application, which can control lights and water for plants based on the light and soil moisture sensor, can be implemented with only 20 lines of JS code.

IV. Evaluation

We implemented an OPEL prototype device with Raspberry-Pi2 board equipped with 900Mhz ARM Cortex-A7 Quad Core and 1GB SDRAM, and measured several performance factors. The average Round Trip Time between OPEL device and smartphone is only 728ms when a UI object with 150KB JPEG image and 51B text is sent. If selective connection is enabled, the time is reduced to a half. The average FPS of four concurrent Full-HD (1080P) recording applications is 28.17. The average latency of synchronous sensor data acquisition is 1.8ms and the average latency of periodic data acquisition is 1ms.

Reference

[1] "IoT.js. A framework for Internet of Things," 2015, retrieved June 14, 2016 from https://github.com/Samsung/iotjs.

[2] "CESANTA. Mongoose IoT Platform," 2015, retrieved June 1, 2016 from https://www.cesanta.com/products/mongoose-iot.

[3] R. LiKamWa et al., "Starfish: Efficient concurrency support for computer vision applications," *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2015