Develop a Universal Remote Using the EasyConnect IoT Management System

Yun-Wei Lin

***Abstract*—Many Internet of Things (IoT) technologies have been used in applications to makes our life more convenient. The *EasyConnect* IoT management system characterizes an IoT device by its "features" (e.g., acceleration, temperature, and etc.) that are manipulated by the network applications. If a network application handles the individual device features independently, then we can write a software module for each device feature, and the network application can be simply constructed by including these brick-like device feature modules. Based on the concept of device feature, brick-like software modules can provide an efficient control mechanism to simply develop a universal remote to control IoT devices.**

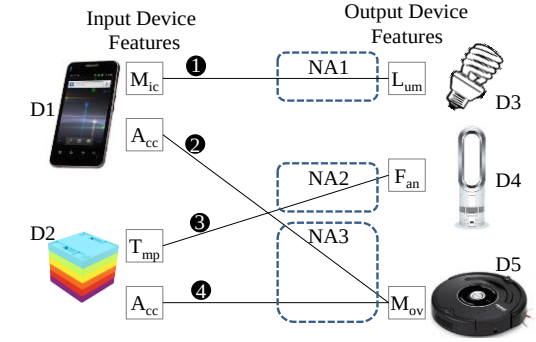
***Index Terms*—Remote controller, internet of things (IoT), machine-to-machine (M2M), wearable device, wireless communications.**

I. Introduction

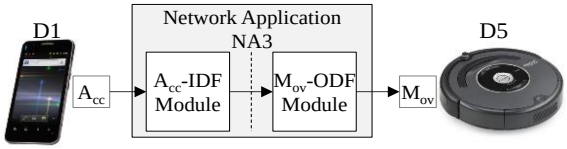
In the recent years, Internet of Things (IoT) has become very popular, which provides interconnection of uniquely identifiable computing devices within the existing Internet infrastructure. The IoT is expected to provide connectivity of devices, systems, and services, which goes beyond machine-to-machine communications with a variety of protocols, domains, and applications [1]. With fast advance of communications and IC fabrication technologies, many new sensors and communication modules for IoT devices have been quickly created for new IoT products.

An IoT device can be characterized by its functionalities or "features". For the purpose of description, this paper defines a feature as a specific input or output "capability" of the IoT device. For example, a smartphone with the accelerator sensor has the input device feature (IDF) called "Acceleration". A blub has the output device feature (ODF) called "Luminance". An IoT device may be connected to the network (i.e., Internet) using wireless communications directly or indirectly. If so, the corresponding software called network application is developed and executed by a server in the network side, which receives or sends the messages from/to the IoT device. When the values of the IDFs are updated, the IoT device will inform the network application to take some actions, and the network application may send the result to the ODF of an IoT device. With this view, the IoT devices interact with each other through their features, and we say that the network application "maps" the IDFs to the ODFs.

Fig. 1 (a) illustrates 5 IoT devices D1, D2, D3, D4, and D5, where the left-hand side of the figure illustrates the IDFs of the devices and the right-hand side of the figure illustrates the ODFs of the devices. The smartphone D1 has two input device features Microphone (M_{ic}) and Acceleration (A_{cc}). The MorSensor [2] D2 has two input device features Acceleration (A_{cc}) and Temperature (T_{mp}). The bulb D3 has an output device feature Luminance (L_{um}). The Fan D4 has one output device feature Fan speeds (F_{an}). The iRobot D5 has one output device feature Movement (M_{ov}) which is used to control the moving direction of iRobot, and its ODF is M_{ov} with two parameters (y_1, y_2) representing movement in two dimensions.



a. Connections among D1, D2, D3, D4, and D5



b. Software module for Acceleration to Movement mapping NA3

Fig. 1. IoT devices, device features, and the network applications.

Lines (1)-(4) in Fig. 1 (a) illustrate how these IoT devices interact, where a line connecting an IDF to an ODF represents interactions between the corresponding device features in input and output IoT devices. Such interactions are implemented in network applications. In Fig. 1 (a), network application NA1 implements interactions (1) for D1 and D3, NA2 implements interactions (3) for D2 and D4, and NA3 implements interactions (2) and (4) for D1, D2, and D5. Consider Line (2) as an example. This line links A_{cc} (the accelerator of D1) to M_{ov} (the Movement of D5), which means that NA3 processes the acceleration value sent from smartphone D1, and then controls the moving direction of iRobot D5.

If a network application handles the individual device features independently, then we can write a software module for each device feature, and the network application can be simply constructed by including these brick-like DF modules. For example, the building blocks for Line (4) in

Manuscript received March 20, 2016; revised December 18, 2016.

Yun-Wei Lin is with the Department of Computer Science National Chiao Tung University Hsinchu, Taiwan, R.O.C. (e-mail: jyneda@gmail.com).

doi: 10.18178/jacn.2016.4.4.229

Fig. 1 (a) are shown in Fig. 1 (b), where the network application NA3 handles Acceleration of D1 and the Movement of D5. This IDF module computes, e.g., the acceleration, and passes the result to the Movement Module. This ODF module translates the received value to the movement direction. Then NA3 outputs this movement direction to drive the movement mechanism of D5. If the IDF and the ODF modules are independent of each other, then these software modules can be reused to build the network applications, and effectively speed up the development of the IoT applications. Fig. 1 (a) shows that different IoT devices may have similar IDFs/ODFs. For example, D1 and D2 have similar input device features Acceleration. Therefore, NA3 can reuse same software modules to implement the tasks for these similar DFs.

Traditionally, a remote is specifically corresponding to a controllable device. In the modern living room, there normally exist several IoT devices which can be controlled by their corresponding remotes, and that causes a lot of remotes are placed around the living room. A user should spend some time to look for a correct remote for the desired IoT device. If there exists a universal remote can control all of the IoT devices in the living room, then the life can be more convenient. In this paper, we show that based on the concept of device feature, brick-like software modules can provide an efficient control mechanism to manipulate IoT devices. Specifically, we develop a multipurpose remote based on EasyConnect [3] to remotely control IoT devices. A user can manipulate a single IoT device, for example, smartphone to remote control home appliances (e.g., TV, air conditioner, lighting, etc.) or other IoT devices through EasyConnect. The user can simply draw a link from an input IoT device to an output IoT device in the EasyConnect GUI (Graphical User Interface), and then she/he can use the input IoT device to remote control the output device.

The paper is organized as follows. Section II discusses the related work. Section III shows the implementation details of this work. Section IV shows how to configure the connections of the appliances/IoT devices through EasyConnect. Section V concludes our work with future research directions.

II. Related Work

Most IoT management platforms, such as Philips hue [4] and IoT.est [5], focus on home automations or sensor networks. Philips hue is a personal wireless lighting system which can only be controlled by a specific smart device (e.g., a smartphone). A smartphone is used as the remote controller to manipulate the colors and luminance of the light bulbs. IoT.est is a test-driven IoT service creation environment, which tests the behavior of IoT devices and services. As an example, consider a motion induction light device (an output IoT device) with a built-in control program. When this program receives a motion event from a motion sensor, it turns on the light. To test this output IoT device, IoT.est simulates the motion detector by sending a motion event to the control program. Then it examines the program execution flow, and checks whether the control program will turn on the light or not. In this way, IoT.est verifies if the control program works correctly. Like Philips hue, IoT.est does not focus on how to connect multiple input IoT devices to multiple output IoT devices. Also, the functions of these platforms are not as modularized and re-useable. Therefore, it is not easy to quickly rebuild a new control mechanism based on the existing software to control other IoT devices.

Arduino Yun [6] is a microcontroller board based on the ATmega32u4 and the Linino AR9331, which provide good communication capabilities to be a proxy between appliances or IoT devices. Appliances, such a fan, an air conditioner, and an iRobot, may only control by IR remote controllers. In this case, these appliances can only communication to each other through a proxy. The IR signals can be generated by ATmega32u4 (Fig.2 (a)) transmitted through an output pin (Fig. 2 (b)) connected with an IR led. ATmega32u4 is suitable to control electronic components and logic circuits. To communication with other IoT devices, Linino AR9331 (Fig. 2 (c)) offers WiFi communication capabilities (Fig. 2(d)) to connect with other IoT devices or communicate with network servers. The embedded Linux OS is running on AR9331, which allows users to develop applications with various programming languages, such as Python. Linino AR 9331 and ATmega32u4 can communicate with each other through the Bridge (Fig. 2 (e)) which provides a data storage function.

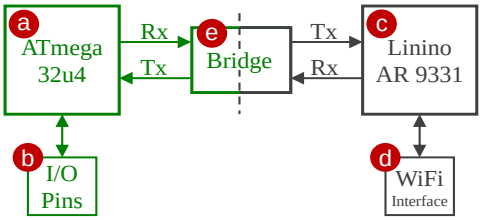


Fig. 2. The bridge communication of Arduino Yun.

III. System Architecture

A. Operating Environment

Assume a smart house with several appliances which can be controlled from networks as shown in Fig. 3. Each appliance connects to an IoT management system installed in the WiFi access point. In this work, we use EasyConnect as the IoT management system. Since the WiFi access point is a very common device in the home environment, a user does not need a new device to execute EasyConnect.

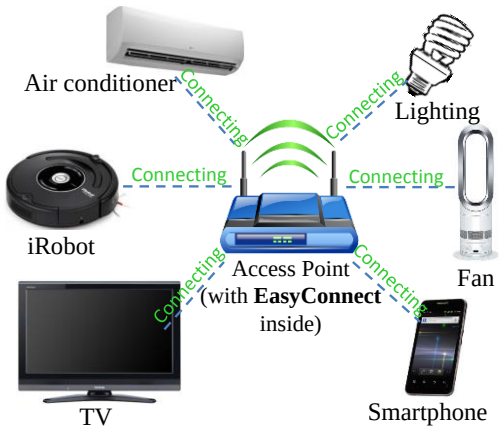


Fig. 3. Operating environment.

B. EasyConnect

EasyConnect is installed and executed on an Intel Edison Breakout Board [7]. Meanwhile, this Intel Edison Breakout Board also acts as the WiFi access point; therefore, all IoT devices naturally connect to this board. Fig. 4 illustrates the EasyConnect architecture. EasyConnect (Fig. 4 (a)) consists of four systems. The *Creation, Configuration and Management* system (abbreviated as the CCM; Fig. 4 (b)) systematically categorizes the features of the IoT devices, manages the functions to automatically configure connectivity of IDFs and ODFs, and stores all related information in the *Database* system (abbreviated as the Database; Fig. 4 (c)). The *Communication SubModule* system (abbreviated as the CSM; Fig. 4 (d)) provides HTTP based RESTful APIs [8] for the *Device Application* system (abbreviated as the DA; Fig. 4 (g)) to deliver/retrieve the IDF/ODF information. When an IoT device connects/disconnects to/from the EC, the DA instructs the CSM through RESTful APIs to change the device status in the EC. In this thesis, the IoT devices are physically connected to the DA, and then transparently communicate with each other through the CSM of EasyConnect. The *Execution SubModule* system (abbreviated as the ESM; Fig. 4 (e)) is responsible for execution of network applications for the connected IDFs and ODFs. The GUI (Fig. 4 (f)) provides a friendly web-based user interface to quickly establish the connections and meaningful interactions among the IoT devices. Through this GUI, a user instructs the CCM to execute desired tasks through CCM procedures to create or set up device features, DF functions, and connection configurations.

The DA (Fig. 4 (g)) is responsible for appliances/IoT devices to communicate with EasyConnect, which is installed in a mobile device (e.g., a smartphone) or an MCU board (e.g., Arduino Yun). It consists of two software components. The Device *Application to the Network* (DAN; Fig. 4 (h)) communicates with EasyConnect for the *IoT Device Application* (IDA; Fig. 4 (i)) registration and data exchange through Wi-Fi. The DA is a Python program which is running on Linino AR 9331. The IDA is an Arduino program and executed on ATmega32u4. The IDA connects to EasyConnect indirectly through the *Device Application to IoT device* (DAI; Fig. 4 (j)). For Arduino Yun, the IDA to the DAI communicates through the Bridge (Fig. 4 (k)).

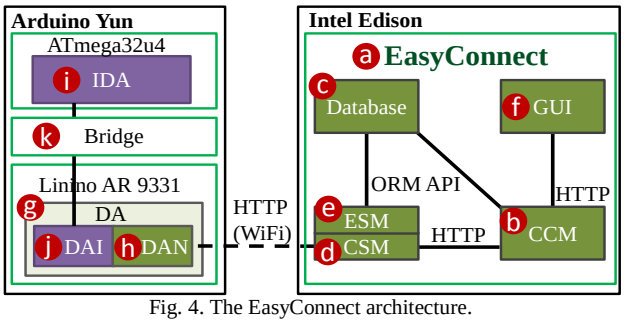


Fig. 4. The EasyConnect architecture.

EasyConnect manages IoT devices with scalability and flexibility, where every IoT device is identified by its *device name* and *device model*. For example, D1 in Fig. 1 is the device name of the smartphone device model. Since different IoT devices of a device model may involve in a connection configuration, device names are needed to distinguish these same-model devices. By considering a device model as a set of device features, EasyConnect effectively manages these device features.

C. Device Applications

The DA is a Python program and executed on Linino AR 9331 of an Arduino Yun board. EasyConnect (Fig. 5 (a)) can control the Arduino program (the IDA; Fig. 5 (b)) on ATmega32u4 through the DA (Fig. 5 (c)) on AR 9331. The DA consists of three components. The DAN (Fig. 5 (d)) communicates with EasyConnect for the DAI (Fig. 5 (e)) through *csmapi* (Fig. 5 (f)). The csmapi is a Python module provided by EasyConnect to assist the DA development. The DA is in charge of four processes, device registration, ODF extraction, IDF decoder, and device deregistration. The details are elaborated as follows.

1) Device registration

The device registration progress (Fig. 5 (g)) is used to register an appliance/IoT device in EasyConnect, where the appliance/IoT device is controlled by the IDA. To register the appliance/IoT device, the function register_device() (Fig. 5 (h)) in the DAN is called. This function then calls detect_ec_ip() (Fig. 5 (i)) to get the IP address of the EasyConnect server. The EasyConnect server periodically broadcasts a string "EasyConnect" on UDP port 17000. The function detect_ec_ip() listens UDP port 17000 and extracts the server IP from this broadcast packet. Once the server IP is obtained, the IP is appended to ENDPOINT of csmapi (Fig. 5 (j)). Then functions in the csmapi know the server IP when the csmapi is called. The function get_mac_addr() (Fig. 5 (k)) is called to get the WiFi mac address. The csmapi uses the mac address as the unique name to register (Fig. 5 (l)) in EasyConnect. The built-in LED on the Arduino Yun (control by pin 13) is turned on by the process Pin13_LED (Fig. 5 (m)) in the IDA to announce the successful registration. Then the appliance/IoT device has been registered in the EasyConnect and can be controlled by network applications.

2) ODF extraction

The ODF extraction process (Fig. 5 (n)) is used to retrieve data in EasyConnect. This process periodically calls the function pull_data() (Fig. 5 (o)) in the DAN to retrieve the data for the appliance/IoT device. The function extract_data() (Fig. 5 (p)) is a JSON parser to extract the required data form a JSON packet retrieved by the function pull() (Fig. 5 (q)). Then the ODF extraction process delivers the data to the output device feature process (Fig. 5 (r)) in the IDA through the Bridge. Then the output device feature process controls the behaviors of the appliance/IoT device based on the received data.

3) IDF decoder

The IDF decoder process (Fig. 5 (s)) decodes the received data from the input device feature process (Fig. 5 (t)). The input device feature process sends the data inputted form Arduino Yun, where the data can be the sensed data from sensors. Then the data are sent to EasyConnect by functions push_data() (Fig. 5 (u)) in the DAN and push() (Fig. 5 (v)) in the csmapi.

4) Device deregistration

The device deregistration process (Fig. 5 (w)) is executed when the appliance/IoT device intends to disconnect from EasyConnect. Functions deregister_device() (Fig. 5 (x)) in the DAN and deregister() (Fig. 5 (y)) in the csmapi are called to disconnect from EasyConnect.

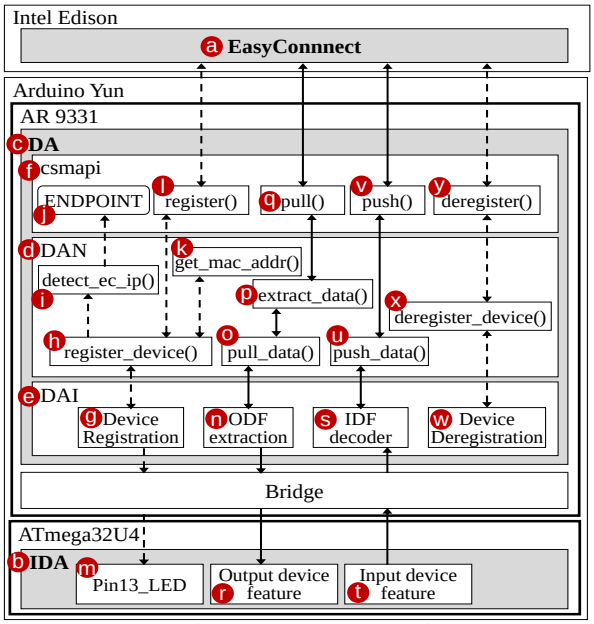


Fig. 5. The functional block diagram of the DA

IV. Configuring the Connections for the Appliances/IoT Devices

This section shows how to dynamically connect IoT devices D2 to D3, D4, and D5 to create the universal remote application. EasyConnect provides a web-based project page (Fig. 6) for a user to connect the device models. The connection configuration of the device models is then saved as a project. When configuring the connection, actual devices need not exist. When the user executes the project (activates the connection), all appliances/IoT devices of the connected device models must register to EasyConnect. Then the input devices can be the remotes to control the out devices.

The left-hand side of the GUI (Fig. 6 (a)) shows the input devices (MorSensor in this case) and the right-hand side (Fig. 6 (b)) shows the output devices. The connection configuration can be set by clicking "Join 1" (Fig. 6 (c)). The details for connection configuration are elaborated in [3]. The user can click the pull-down menu "Model" (Fig. 6 (d)) to add the device models MorSensor (Fig. 6 (e)) and iRobot (Fig. 6 (f)). Then the user clicks the input device feature Acceleration of MorSensor (Fig. 6 (g)) and the output device feature Movement of iRobot (Fig. 6 (d)) to draw a link. Then the user can use the acceleration of MorSensor to control the movement direction of iRobot. Through the similar way, the user can create the link between MorSensor and Fan, TV, or light to control them, and the MorSensor becomes the universal remote.

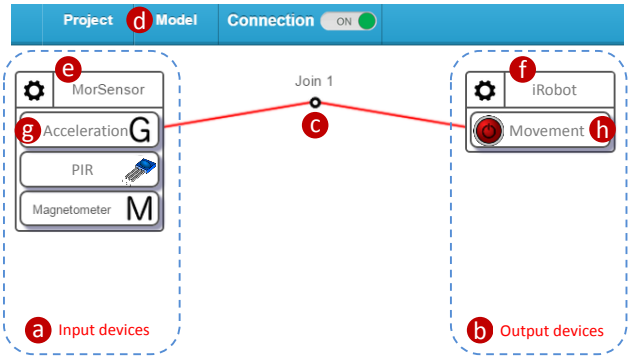


Fig. 6. The GUI for connection configuration.

V. Conclusion

This paper described the design and implementation of universal remote using EasyConnect, a device feature management system that effectively and flexibly links the IoT devices. In our approach, an IoT device is characterized by its "features" (e.g., temperature, vibration, display, etc.) that are manipulated by the network applications. By considering a device model as a set of device features, EasyConnect effectively classifies these device features, and manipulates them through network applications. Therefore, a user can manipulate a single IoT device, for example, smartphone to remote control home appliances (e.g., TV, air conditioner, lighting, and etc.) or other IoT devices through EasyConnect. The user can simply draw a link from an input IoT device to an output IoT device in the EasyConnect GUI, and then she/he can use the input IoT device to remote control the output device.

References

[1] J. Höller, V. Tsiatsis, C. Mulligan, S. Karnouskos, S. Avesand, D. Boyle, *From Machine-to-Machine to the Internet of Things: Introduction to a New Age of Intelligence*, Elsevier, 2014.

[2] MorSensor 2016. [Online]. Available: http://www.narlabs.org.tw

[3] Y. B. Lin, Y. W. Lin, C. Y. Chih, T. Y. Li, C. C. Tai, Y. C. Wang, F. J. Lin, H. C. Kuo, C. C. Huang, and S. C. Hsu, "Easyconnect: a management system for IOT devices and its applications for interactive design and art," *IEEE Internet of Things Journal*, no. 99, 2015.

[4] Philips hue (2016). [Online]. Available: http://www.developers.meethue.com/

[5] S. De, F. Carrez, E. Reetz, R. Tonjes, and W. Wang, "Test-enabled architecture for iot service creation and provisioning," *The Future Internet Lecture Notes in Computer Science*, vol. 7858, pp. 233-245, 2013.

[6] Arduino Yun. (2016). [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardYun

[7] Intel Edison. (2016). [Online]. Available: http://www.intel.com/content/ www/us/en/do-it-yourself/edison.html

[8] Representational state transfer (REST). (2016). [Online]. Available: https://en.wikipedia.org/wiki/Representational_state_transfer

Yun-Wei Lin received the B.S. degree in computer and information science from Aletheia University, Taipei, Taiwan, in June 2003, and the M.S. and Ph.D. degrees in computer science and information engineering from National Chung Cheng University, Chiayi, Taiwan, in 2005 and 2011, respectively. His current research interests include mobile ad hoc network, wireless sensor network, vehicular ad hoc networks, and M2M communications.