DOI: 10.3969/j.issn.1673-5188.2017.04.002
http://kns.cnki.net/kcms/detail/34.1294.TN.20171016.1814.002.html, published online October 16, 2017

An OS for Internet of Everything: Early Experience from A Smart Home Prototype

CAO Jie, XU Lanyu, Raef Abdallah, and SHI Weisong
(Department of Computer Science, Wayne State Unversity, Detroit, MI 48201, USA)

Abstract

The proliferation of the Internet of Everything (IoE) has pulled computing to the edge of the network, such as smart homes, autonomous vehicles, robots, and so on. The operating system as the manager of the computing resources, is also facing new challenges. For IoE systems and applications, an innovative operating system is missing to support services, collect data, and manage the things. However, IoE applications are all around us and increasingly becoming a necessity rather than a luxury. Therefore, it is important that the process of configuring and adding devices to the IoE is not a complex one. The ease of installation, operation, and maintenance of devices on the network unarguably plays an important role in the wide spread use of IoE devices in smart homes and everywhere else. In this paper, we propose Sofie, which is a smart operating system for the IoE. We also give the design of Sofie. Sofie can be implemented via different IoT systems, such as Home Assistant, openHAB, and so on. In order to implement Sofie to get some early experience, we leverage Home Assistant to build a prototype for the smart home. Our work shows that Sofie could be helpful for practitioners to better manage their IoE systems.

Keywords

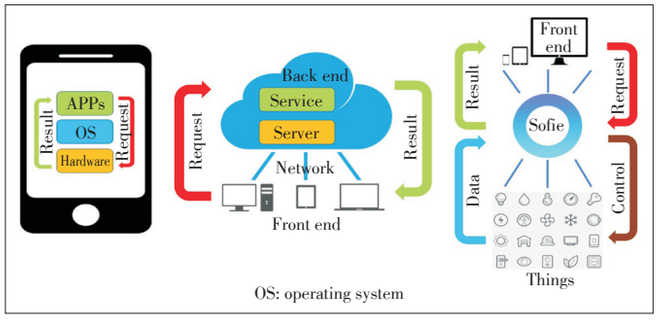
edge computing; IoE; smart home; operating system

1 Introduction

With the burgeoning of the Internet of Everything (IoE), computing in our lives is shifting from PC, mobile device, and cloud to the things at the edge of the network [1]. More and more data is generated as well as consumed at the edge of the network. According to the report from Cisco Global Cloud Index, the data produced by people, machines, and things will reach 500 zettabytes by year 2019 [2]. Moreover, 45% of the data contributed by the things will be stored, processed, analyzed, and acted upon close to, or at the edge of the network [3]. While more devices and applications are coming out for IoE, the operating system (OS) for IoE is still unavailable. Conventional operating systems for PC and smart phones care more about resource management. The application practitioners directly use hardware resources since the hardware of a computing platform is well-designed and forms a fixed environment. For cloud computing, a service provider will manage all the hardware so that application practitioners only need to focus on the service. In this case, cloud computing OS is designed as a service-oriented architecture, and various members of the "as a Service" (aaS) family such as Software as a Service (SaaS), Infrastructure as a Service (IaaS), and Platform as a Service (PaaS).

The IoE operating system should have similar functions as traditional operating systems; however, there are several differences. In the IoE, similar as cloud computing, application practitioners should not care about the hardware. However, unlike cloud computing system that has service-oriented architecture, an IoE operating system should be data-oriented, considering that most of the things are just passively collected and the reported data are in a predefined manner. Moreover, IoE operating system should also be responsible for hardware management like PC or smart phone operating systems. This is because the IoE is a highly dynamic system where devices are added or removed frequently.

Therefore, traditional OS is not suitable for the IoE. In this paper we introduce a smart operating system for the IoE—Sofie. In **Fig. 1**, we compare the mobile operating system, cloud, and Sofie. In the case of mobile device and PC, the operating system focuses more on resource management and provides interface for applications to access hardware resources since the software and hardware are all fixed in a determinate machine. When the applications want to access any hardware resource, they can directly send a request to the operating system. On the other hand, in the case of cloud computing, the users on the front end might have limited information about hardware re-



▲Figure 1. Comparison of mobile operating system, cloud, and Sofie.

sources on the other end of the network. Therefore, the cloud service providers should access the request and send a computing result back to the users in a distributed manner. For IoE, Sofie faces new challenges [4]–[8]. Unlike the mobile, PC, and cloud, where the hardware configuration is fixed to a certain extent, the things in IoE could be highly dynamic and unreliable. Moreover, the things, which compose the back end of the paradigm, usually have very limited computing resources (just passive report data and receive commands). To better serve the users on the front end, Sofie should be able to abstract the data from the things and satisfy the front end by taking the computing back to Sofie, rather than forward the request to the things on the back end.

Sofie is both data-oriented and things-oriented. The detailed design of Sofie will be presented in the following several sections. In this paper, we would like to take smart homes as an example to show how to implement Sofie in a domestic environment and how Sofie works to help practitioners better manage their IoE environments.

We organize the remainder of this article as follows: In Section 2, we present the design of Sofie with things management and data management as two major functions. We further show how Sofie looks like in a real IoE environment. We choose a home as an environment for Sofie and present the architecture of Sofie in Section 3. In Section 4, we review both commercial and open source products for smart homes, and evaluate them from different aspects. We introduce the implementation of Sofie on top of Home Assistant and also show how Sofie works for smart homes in Section 5. In Section 6 we present the lessons learned through our work experience with smart homes. Finally, the paper concludes in Section 7.

2 Design of Sofie

In the IoE, although there are various kinds of sensors and devices that can produce data, they work in a passive manner of data reporting, without adopting any action to process the data [9]–[11]. However, for practitioners, the generated data could be leveraged for multipurpose use in the society. Such utilization fields include video analysis, smart homes, smart cities, connected health, and more. All these fields treat data as an indispensable ingredient. For example, data produced in a smart home is consumed to improve the user experience for the occupants; data captured by traffic cameras is retrieved to track suspicious vehicles; data provided by the police department or city hall is utilized to benefit the public; data collected in connected health is used to facilitate communication among hospitals, pharmacies, insurance companies and so on.

Given the significance of data in IoE applications and systems, as well as the underlying hardware, a data-driven and thing-driven operating system is required in the IoE. In this paper, we propose Sofie, which is a smart operating system for the IoE. As shown in **Fig. 2**, Sofie is sitting between devices and services, as both a service provider of the upper layer and a hardware manager for underlying devices, to provide high quality data through well performed things. For the IoE, Sofie is the brain that manages data, devices, and services. For service practitioners, Sofie is capable of reducing the complexity of development by offering an abstracted data access interface. In regard to functionality, Sofie is divided into two layers: the things management and data management.

The general architecture design is shown in **Fig. 3**. On the one hand, Sofie has the capability of maintaining all the connected things, since most of the involved things just work passively to generate data, while their number is too large to be tracked manually. On the other hand, Sofie is designed to encapsulate underlying things well, and only provide the data access interface, since super-stratum services only care about the data and do not need to know the low-level status. In the remaining part of this session, we will introduce Sofie from the views of things and data respectively.

2.1 Things Management

Taking control of different kinds of devices and sensors, Sofie has responsibility to reduce human intervention work, and coordinate all the devices to guarantee stable functionality and performance. In order to support qualified performance, the things management layer is made up of two parts: things config-

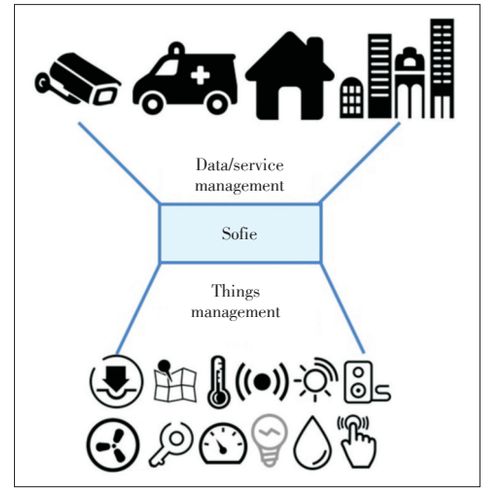
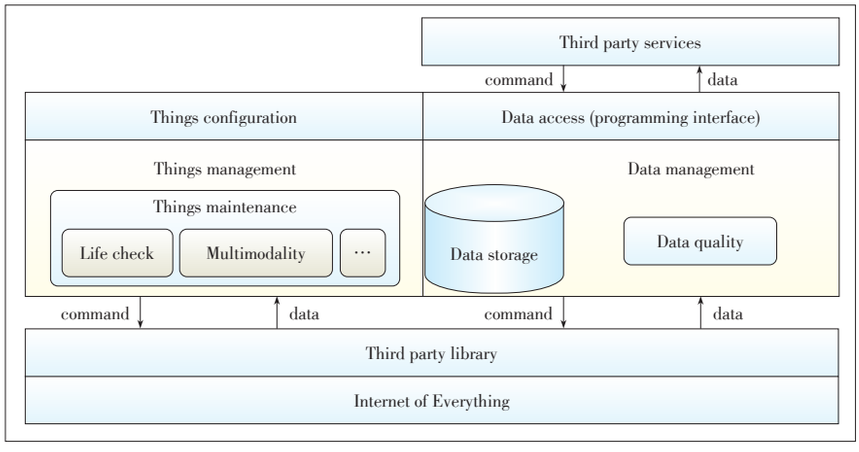


Figure 2. ▶ Overview of Sofie.



▲Figure 3. Architecture of Sofie.

uration and things maintenance.

2.1.1 Things Configuration

Things configuration includes things registration and replacement. Although manual operation is inevitable in this part, Sofie takes care of all the remaining work after the device is connected to the system. When a new device is connected to the system, Sofie searches configuration files for its available service. Meanwhile, the identity information of a new device is recorded for Sofie to distinguish the new added device from other devices.

If a device dies or does not perform well, Sofie would raise a replacement request. Before the replacement happens, a compensation method is needed to preserve the stable performance and avoid any service interruption or damage to the system. In a relatively small and closed environment, suspending all services related to the malfunctioning device is a practicable method. In a relatively large and open environment, where services are more complex and mature, one alternative way is assigning the tasks of the malfunctioned device to adjacent devices temporarily. After the replacement is complete, Sofie will restore the states of the malfunctioning device to the new one, including the related services or functions.

2.1.2 Things Maintenance

Given the complexity and specificity of the IoE, it is not feasible for a human to take care of all the devices [12]–[16]. Thus, things maintenance is actually the mechanism of self-management for Sofie. When there is no human intervention, Sofie is the dominator to coordinate all the devices and sensors.

There are several ways to monitor health status of things. In this paper, we propose the following two techniques that will be implemented in Section 5.The general method is life checking. For each device connected to Sofie, a heartbeat signal is required to be sent to the system in regular time intervals. If no heartbeat is received from a certain device, Sofie could figure out the device is dead or disconnected from the system, and a relevant compensation method could be raised. In addition to life checking, which is seen as the regular monitoring method, some unconventional monitoring ways would be very helpful in extraordinary situations. For instance, when a device keeps sending heartbeats, while sending irregular data to the system compared to previous records, it can be reasonably inferred that there is something wrong with that device. Multimodality checking is another way to monitor the health status of things in Sofie. It harmoniously combines the different capabilities of connected devices as an auxiliary means to check the possible failure of a certain device. Smart home is one scenario that could implement this method. Data could be used here including video/image information recorded by cameras, temperature information recorded by temperature sensors, sound information recorded by acoustic sensors, etc. Correspondingly, multimodality checking could be adopted between a camera and a light, an audio system and an acoustic sensor, an air conditioning system and a temperature sensor and so on. To be specific, if a light component is not operating normally, Sofie could invoke domestic security camera to check if the light bulb works when its service is switched to on mode. The camera could take a picture or a short video of the surrounding environment of the light, and send it to the system for analysis. Similar to life checking process, if a device is observed to perform poorly, relevant remedial actions will be initiated. These actions could call neighboring devices to temporarily share the task of the broken device, or ask for a replacement.

2.2 Data Management

As a things-driven and data-driven operating system, data is another criterion to Sofie. Data management contains not only data quality, but also data storage and data access.

2.2.1 Data Quality

As we discussed before, the IoE is a broad concept, including different kinds of hardware that anyone can think of. Things in it could be as small as a motion sensor, or as large as a city. Due to the limitation of computational capacity, energy capacity and so on, most of the things only produce data in a passive way. Due to unstable wireless connection and volatile environment, the IoE is fragile to some extent. Considering the unreliable environment of the IoE, it is important that the data is of high quality, which means the data should be valid, complete, and timely. In general, the data quality of Sofie could be evaluated by two criteria: history pattern and reference data.

Data easily form a certain pattern due to the periodical activity of human and nature. To detect abnormal data and provide high quality data, historical data records could be employed by

some data mining algorithms. If newly acquired data differ significantly from the history pattern, Sofie would analyze the reason for that, which could be device failure, communication interface change, or hostile attack.

In some cases, it is unavoidable that some devices or sensors malfunction. For example, in the case of several traffic cameras over road junctions where one traffic camera malfunctions and keeps sending low quality data, the data processed by nearby cameras could be leveraged as the reference before a replacement is complete.

2.2.2 Data Storage

Data storage is integral to Sofie data management. Underlying devices of Sofie generate a large variety of data every day. Some of the data is just redundant, while some could be valuable for future use. To cater the design that service should be isolated from things, valuable data should be abstracted before being stored in the database.

The database in Sofie is used to record the following four kinds of information:

- Device information. When a new device is added to Sofie, the identity information is recorded, as well as some con- figuration related information, such as related services. When a replacement happens, this kind of information could be used to restore previous configuration.
- State history. Basically, the state of a device could be divided into three cases: on, off, and unreachable. Both on and off indicate the device is available to the system, while unreachable state shows the device is disconnected. State information is significant for Sofie to manage things.
- Event history. It records every action of a device triggered by a related service. Event history would be useful when analyzing the history pattern of a certain device, or retrieving an event at a given time.
- File path. This is an alternative information, specifically designed for systems that include cameras. For such systems, video or image storage usually consumes a significant amount of space. Therefore, storing file paths in database is meaningful for secondary development.

2.2.3 Data Access

Without a uniform programming interface, developers would spend considerable time and effort to get data from different devices. Created for upper layer service, a programming interface is developed to make data access in the IoE flexible and convenient to service practitioners, as well as a tool against potential malicious attacks. Utilizing the database, the developer is able to utilize the unified interface to get abstracted data from Sofie.

3 Sofie at Home

In this section, we present the home environment as an example for Sofie, show how to design and implement a home OS based on Sofie, and address the widespread latency challenge in smart home environment.

The design of Sofie at home is shown in **Fig. 4**, including seven components: the communication adapter, event hub, database, self - learning engine, application programming interface, service registry, and name management that stretches across other components.

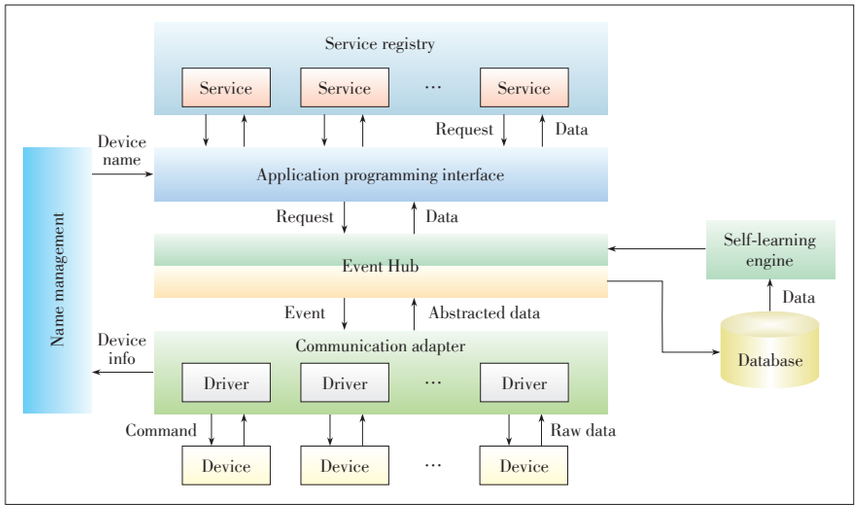
To integrate a device into Sofie, the communication adapter gets access to the device by the embedded drivers. These drivers are responsible for sending commands to devices and collecting state data (raw data) from them. Sitting between devices and the event hub, the communication adapter packages different communication methods that come from various kind of devices, while providing a uniform interface for upper layers' invocation. In this way, developers and users do not need to deal with multiple kinds of communication methods when manipulating the system. Moreover, it only provides abstracted data to upper layer components, reducing privacy risk to some extent.

As the core of the architecture, the event hub maps to two layers in the logical view: the data management and self-management layers. The event hub is responsible for capturing system events and sending instructions to lower levels. Those instructions are smart commands based on machine learning developed through communication with the self-learning engine. It collects requests from services and sends them to the communication adapter, and in turn, collects abstracted data from the communication adapter and sends them to upper layers. The database is another component in the data management layer. As a data-oriented system, Sofie generates large amount of data every day, which contains valuable information related to user preferences and settings. The event hub stores data in the database. The data stored in the database is utilized by the self-learning engine that belongs to the self-management layer. The self-learning engine creates a learning model. This learning model called the self-learning model acts as an input to the event hub to provide decision - making capability. To provide better user experience, the self-learning engine is developed to analyze user behavior, generate the personal model for the user, and help improve the system.

The application programming interface (API) and service registry are located in the upper layers of the system, and are utilized by third-party services. Developers are encouraged to use Sofie APIs to communicate with the event hub, and register their services with the system.

Required by all the layers, the name management module helps the system keep devices organized. When a new device is registered with the system, this module allocates a name for it using the following rule: location (where), role (who), and data description (what). This rule is complied by all the layers.

The design of Sofie is quite flexible to accommodate multiple functionality. Recently, with the cognitive technique development, there is a growing market for voice-controllable smart home products like Amazon Echo [17], Google Home [18], and



▲Figure 4. Design of Sofie at home.

Apple HomePod [19]. Such audio-based function is able to be realized in Sofie as a registered service, with the presence of suitable Natural Language Processing (NLP) model [20].

4 Smart Home Systems

There are many systems that can be used to build a smart home. We did not build Sofie from scratch but tried to utilize available systems to fulfill our concepts. In this section, we will review both commercial and open source products for smart homes.

4.1 Commercial Products

With the proliferation of the high-speed Internet and IoE, more and more products for the smart home are also available on the market. A smart device such as iRobot, Philips Hue, and Nest learning thermostat shows that homeowners are ready to embrace smart devices in their daily lives. Amazon Echo, Samsung SmartThings, and Google Home provide a hub and user interface for occupants to interact with connected devices. HomeOS from Microsoft and HomeKit from Apple enable a framework for communicating with and controlling connected accessories in a smart home.

Although there are a bunch of choices on the market to implement a smart home, the lack of proprietary software APIs and competitors' product support makes it hard to use commercial smart home systems in our project.

4.2 Open Source Systems

Despite the commercial products on the market, there are several communities that worked on open source projects for home automation systems. In this section, we compare the popular open source systems that can be used in a smart home, and hope this information could be helpful for the practitioners who plan to leverage open source home automation system in their own work.

In **Table 1**, we compared six open source systems from various aspects including programming languages and documentation support. Different object-oriented programming languages are used in these systems such as Python, Java, C++, C#, and Perl. We also inspected the lines of code for each system. We found out that all the systems' lines of source code fall into the similar magnitude between 100,000 to 1,000,000. Most of the systems except MisterHouse [21] use HTML to provide an interface to interact with the end user on the front end. Some of the systems also developed front end native mobile applications for iOS and Android such as Home Assistant [22], openHAB [23], and HomeGenie [24]. Becides MisterHouse, all of the other systems offer an API for the practitioners to utilize their data and functions. For the data storage, Home Assistant, Domoticz [25], and HomeGenie use SQLite database. OpenHAB and Freedomotic [26] provide data persistence service to the user, which means the user can freely implement customized database through the offered interface. In MisterHouse, the history data is not stored, and the users need to code directly into the source file if they want to implement automation methods. Meanwhile in all of the other systems, automation can be implemented through scripts file. Moreover, for Home Assistant, openHAB, and Freedomotic, automation could also be easily implemented by offering "trigger-condition-action" rules. All of the systems can be running on multiple platforms such as Linux, Windows, and Mac OS.

In the previous sections, we talked about our design of device abstraction and data abstraction. During the review, we found out that none of the current open source systems offered

▼Table 1. Comparison of smart home systems

	Language	Lines of code	Frontend	API	Data storage	Automation	Platform	Device abstraction	Data abstraction	Documentation
Home Assistant	Python3	213,901	HTML, iOS	Y	SQLite	Rules, Scripts	Linux, Win, Mac OS X	Y	N	Good
openHAB	Java	904,316	HTML, Android, iOS, Win	Y	Persistence services	Rules, Scripts	Any device with JVM	Y	N	Good
Domoticz	C++	645,682	HTML	Y	SQLite	Scripts	Linux, Win, Mac OS X	N	N	Poor
Freedomotic	Java	159,976	HTML	Y	Data persistence	Rules, Scripts	Any device with JVM	N	N	Good
HomeGenie	C#	282,724	HTML, Android	Y	SQLite	Scripts	Linux, Win, Mac OS X	N	N	Average
MisterHouse	Perl	690,887	NA	N	NA	Perl code	Linux, Win, Mac OS X	N	N	Poor

API: application programming interface JVM: Java virtual machine OS: operating system

data abstraction function, which means they only store the raw data reported by the devices. Only Home Assistant and openHAB implement device abstraction. They can isolate devices from services by abstracting the role of devices. In this way, services do not need to know the physical information of devices such as IP and MAC addresses.

Documentation is the last index we used to evaluate the systems. A well written and maintained document is extremely helpful for the practitioners to install, configure, and manage the system. Home Assistant, openHAB, and Freedomotic provide good documents for the users.

Based on our evaluation of the above open source systems, we decided to use Home Assistant and openHAB in our project since their architectures fit our design better. We eventually chose Home Assistant over openHAB since Python has less overhead than Java, and the Home Assistant community is more active with updates and latest device support.

5 Implementation

In this section, we will introduce the architecture of Home Assistant and the implementation of Sofie on top of it.

5.1 Architecture of Home Assistant

Fig. 5 shows the architecture of Home Assistant. For the things with open APIs (as most of the products on the market are), there are usually some third party libraries written in Python by the community. Home Assistant utilizes those libraries and abstract things into different components. Then on top of the components is the core of Home Assistant, where a state machine, an event bus, and a service registry are supported to control and manage the components. A user interface is offered on top of the Home Assistant core for users to access the home information and control devices. Home automation is also supported by providing a YAML (YAML Ain't Markup Language) [27] configuration file.

5.2 Sofie on Top of Home Assistant

Based on the comparison of open source smart home systems, we chose the communication layer of Home Assistant as the substructure, and developed Sofie on top of it. As we claimed in the previous section, Home Assistant has implemented device abstraction in a comprehensive way. More than 600 components are supported by Home Assistant, and universal interfaces are provided for different kinds of devices. We use Home Assistant as an open source package to build Sofie prototype because Home Assistant solves the driver issue, which facilitates the creation of Sofie, and avoids redundant work.

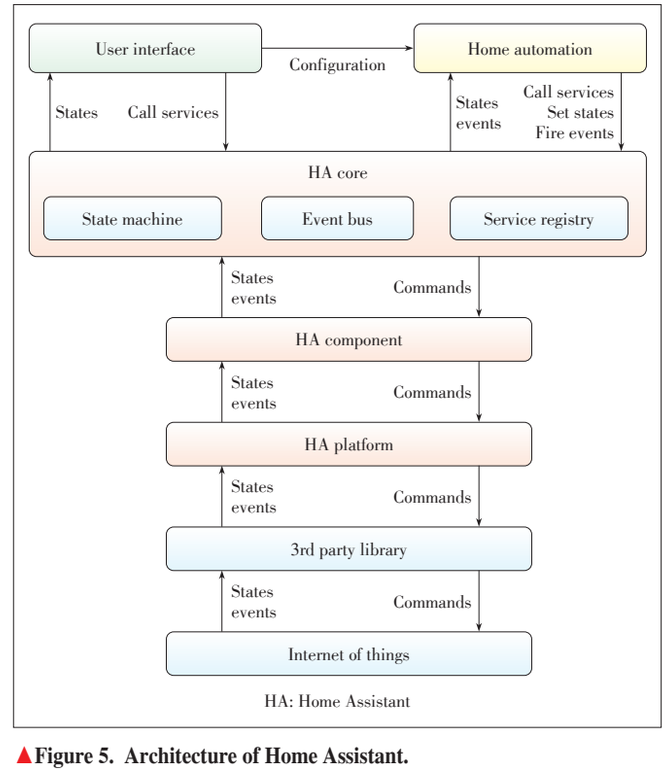
To successfully implement functions required by Sofie, we modified the components in the event bus layer and service layer, as shown in **Fig. 6**. In comparison with Home Assistant, which focuses on the functionality of the system, Sofie pays attention to both functionality and the self-management of devices. To demonstrate Sofie's capability of data management and things management, we conducted several preliminary experiments by leveraging Home Assistant framework.

5.2.1 Database Renovation

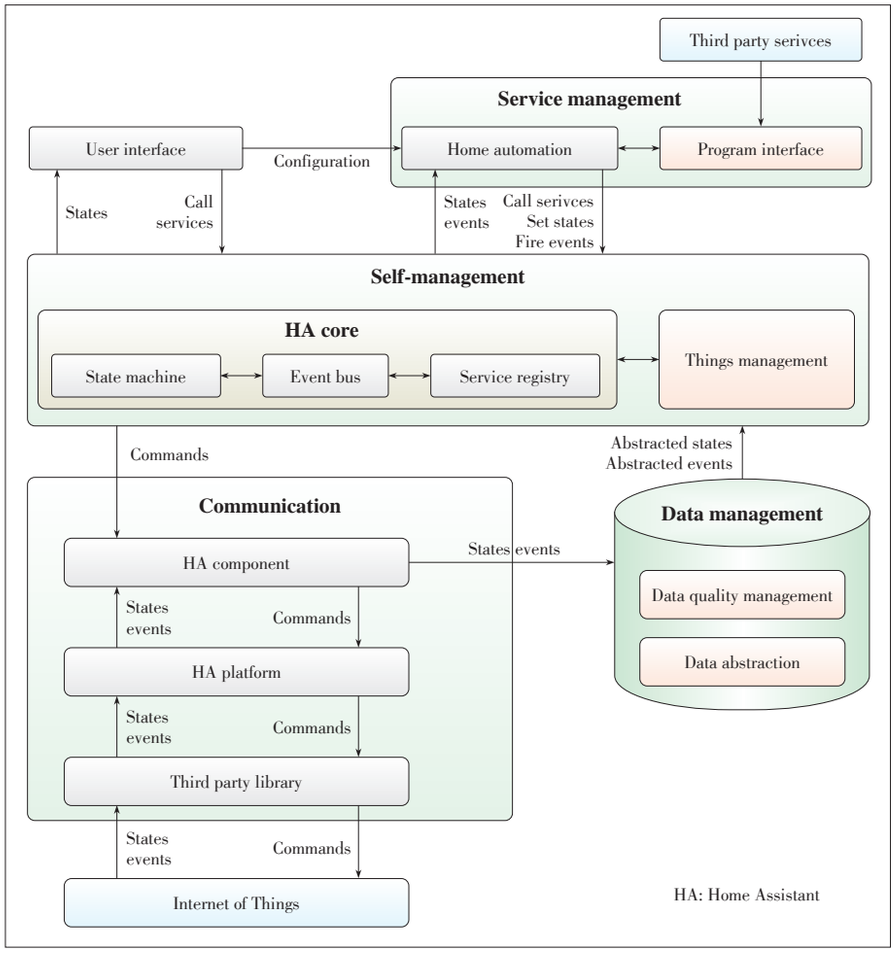
Home Assistant uses SQLite as the default back end database to store historical data. There are four tables in the database:

- The events table records what type of event happens of what device and at what time. It is frequently updated once a service is registered, a new platform is connected, or a device state is changed, and the detailed information will be stored in this table.
- The recorder runs table records the start and end time of the whole system. It only updates when Home Assistant starts or ends.
- The schema changes table records the system update history. It seldom receives and stores new data compared to other tables.
- The states table records changes in device status, including the last changed time and last updated time. It is an active table since the device state changes from time to time in the smart home environment.

To successfully realize the functionality of Sofie, we added the following two tables. One is the IP address table to store the IP addresses of registered devices. It updates when new devices are added to the system, when a replacement occurs, or



▲**Figure 5. Architecture of Home Assistant.**



▲Figure 6. Architecture of Sofie on top of Home Assistant.

when a connected device changes the IP address. The life checking component relies on this table to check the reachability of registered devices. The other is the file path table to store the image/video files recorded by security cameras. Video images are important information for both Sofie and top layer services. For Sofie, this information could be utilized in device management. More details will be discussed in 5.2.4. Services built on top of Sofie may need to fetch historical image data to finish the tasks. For instance, this table could provide the path of related files to a service by analyzing historical data when the service needs to track a specific person or an event. The update frequency of this table depends on the setting and could vary greatly. In our experiment, it is only updated when a certain event occurs.

5.2.2 Life Checking

Based on the Sofie design discussed in Section 2, monitoring the healthy status of devices could help maintain system performance. The general method to track the devices is life checking. Considering the passive character of most devices, the core is set as the action initiator to start the life checking. In our configuration, all the IP addresses in the database are attempted in a round-robin manner by the system every five minutes. If an IP address could not be accessed, Sofie will suspend its current configuration and mark the state as "unreachable" both in database and user interface. Otherwise, the working state will be displayed on the user interface. Generally speaking, "unreachable" is not necessary because of device failure just because the user turned off the device through its physical switch, or due to environment power outage.

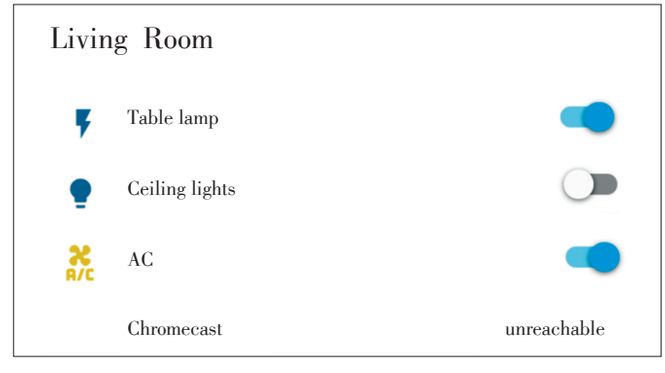
Fig. 7 is an example of a user interface with life checking function. There are four devices in the living room: a table lamp controlled by a smart switch, ceiling lights composed of several smart light bulbs, a smart thermostat, and Chromecast. During the life checking round, the Chromecast is inaccessible, and therefore the user interface displays "unreachable" as its current state.

5.2.3 APIs

Sofie provides two APIs for practitioners to work with: a Python API inherited from Home Assistant to interact with things, and a database application programming interface (DB-API) for SQLite database. With the Python API shown in **Fig. 8**, developers can connect to Sofie remotely with IP address and password combination, get data from Sofie, and also control the devices. With DB-API shown in **Fig. 9**, developers can connect to the Sofie database directly if the service only cares about the home data.

5.2.4 Multimodality Checking

Another way to check the healthy status of devices is multi-



▲Figure 7. Home Assistant user interface with life checking.

```
# Python-API interface for Sofie inherited from HA
import homeassistant.remote as remote

# Connect to Sofie Hub
api = remote.API('192.168.0.1', 'YOUR_PASSWORD')

# Get available services, events, and entities
services = remote.get_services(api)
events = remote.get_event_listeners(api)
entities = remote.get_states(api)
```

Figure 8. Python API example code for Sofie.

```
# DB-API interface for SQLite databases
import sqlite3

# Connect to database
comm = sqlite3.connect('Sofie.db')
c = conn.cursor()

# Query data from database
c.execute("SELECT * FROM event_data WHERE
entity_id='switch.livingronn_switch'")

# Save (commit) the changes
conn.comit()

# Close the connection
conn.close()
```

Figure 9. DB-API example code for Sofie.

modality checking. When a new device is registered, the user is allowed to set up a rule for multimodality checking. The rule manner is *Name_of_device1: multimodal: name_of_device2*. This rule states that Device 2 will be invoked when Sofie checks the healthy status of Device 1. Such status check relationship could exist in different kinds of devices such as light and camera, AC system and temperature sensor, and sound system and acoustic sensor. In the preliminary experiment, we used light and camera as an example, in which the security camera plays a key role in taking pictures and utilizing vision related libraries to process the captured images. Assuming a relatively dark background, Sofie checks if the light is operating normally by triggering the camera to take a picture when the light turns on and compare it with a previously saved one when the light was off. Then Sofie retrieves the vision related library to compare the difference between the two images to check if the light successfully turned on.

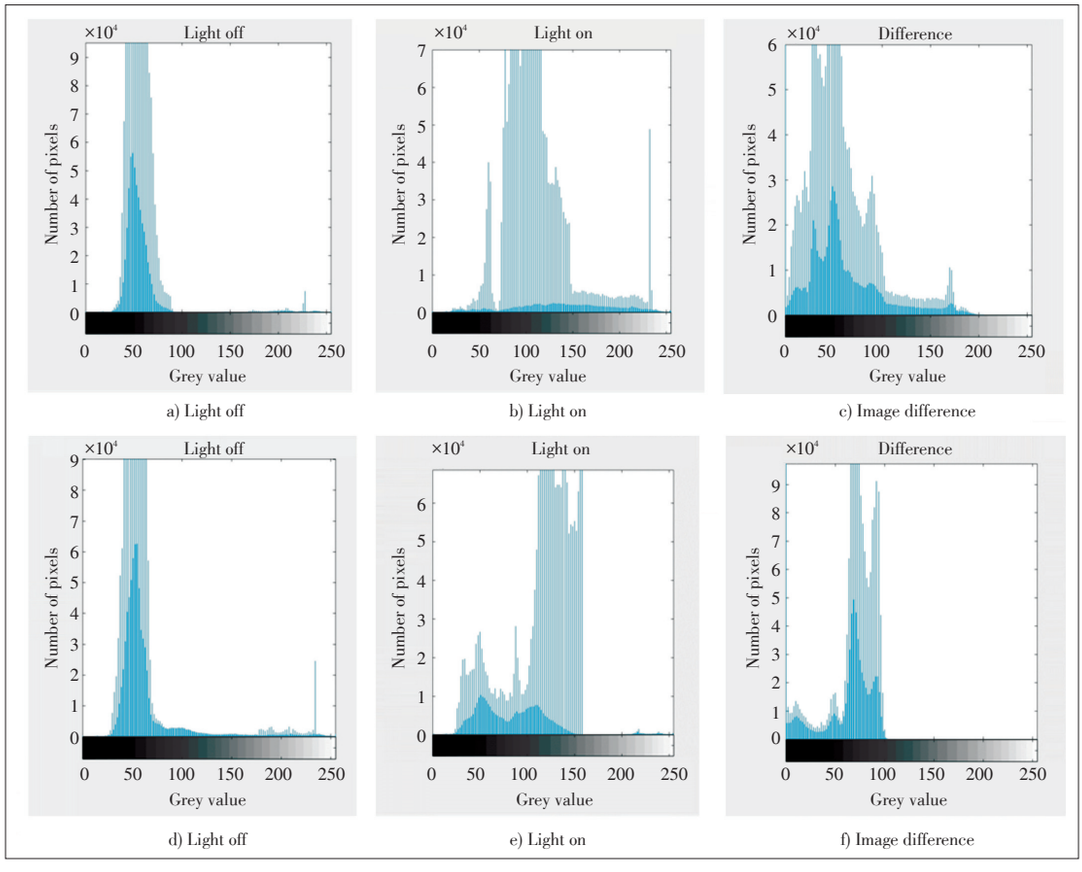
Figs. 10 and **11** show the multimodality checking process. The pictures in Figs. 10a and 10b are taken from the same angle, and the same applies to Figs. 10c and 10d. The generated gray level histogram of each picture is shown in Fig. 11. Figs. 11c and 11f are the gray level histogram of difference image of Figs. 10a and 10b, and Figs. 10c and 10d respectively. To analyze the grey level images, an appropriate threshold method is meaningful [28]. As a preliminary experiment, we set the threshold of pixel gray level to 50. That is, if more than half of the pixels are larger than 50 in the image difference gray level histogram, the system treats the light as turned on. The same method standard can also be used to confirm the light is turned

Figure 10. Camera snapshots when light turns on/off.

off. During the preliminary experiment, the average response time is 220 ms for a complete multimodality checking process, which including taking picture, transferring message, comparing two pictures, and getting the conclusion. Multimodality checking is not limited to monitoring a light's state. In a mature smart home running Sofie, the security camera has the ability to check several devices such as a smart stove and a washer machine.

5.2.5 Configuration File

When Sofie starts, it will reach out to the configuration file to set up the initial smart home environment. Inherited from Home Assistant, the configuration file is consisted of two parts: 1) default settings, including the user interface (username and password) and geographic location (longitude, latitude, time zone, etc.); 2) custom settings input by the user, telling the system what kind of device is ready to connect and what kind of



◀Figure 11. Gray level histogram corresponding to the snapshots in Fig. 10.

service is required. A sample configuration file is shown in **Fig. 12**. Usually, same kind of devices are grouped to the same component, then distinguished by different platforms (brands) and IP addresses. Fig. 12b defines one media player (Roku TV), one light group (control through Phillips Hue), and two security cameras (Amcrest camera, and MJPEG camera). According to this configuration file, the living room camera is involved in the multimodality checking of light.

In this section, we introduced how we implement Sofie for a smart home, and we also showed how to implement our design on the top of an open source system. An open source package of Sofie is going to be released on our website soon.

6 Discussion

Here, we share some lessons and experience we have learned during the design and development of a smart home.

6.1 Latency in Sofie at Home

In a smart home, the response time is an important metric to determine whether a smart home system is satisfactory or not [29]– [35]. Sofie is no exception. Back in 1968, Miller described computer mainframe responsiveness in three different orders of magnitude [36]: 1) 100 ms is perceived as instantaneous; 2) 1 s or less is fast enough for the user to interact with the system in a free way; 3) 10 s or more reduces the user's interest. Generally speaking, controlling the response time under 1 s is sufficient for the satisfactory functionality of the smart home implementation, and meeting user expectations.

Fig. 13 shows the latency of turning on light in both conven-

```
homeassistant:
  # Name of the location where Home Assistant is running
  name: Home
  # Location required to calculate the time the sun rises and sets
  latitude: 42.3463
  longitude: -83.0598
  # Impacts weather/sunrise data (altitude above sea level in meters)
  elevation: 187
  # metric for Metric, imperial for Imperial
  unit_system: imperial
  # Pick yours from here: http://en.wikipedia.org/wiki/List_of_tz_database_time_zones
  time_zone: America/Detroit
```

a) Default settings

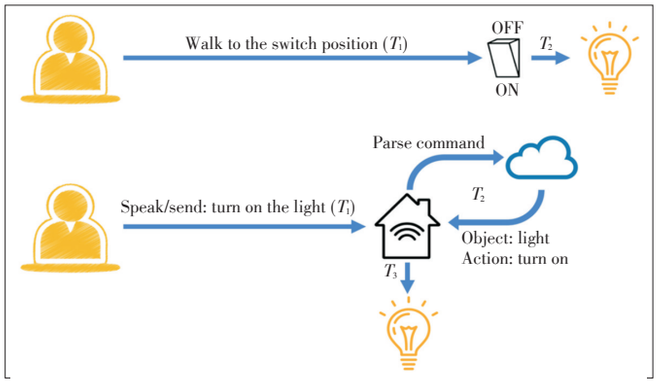
```
media_player:
  - platform: roku

light:
  platform: hue
  host: hue_IP_address
  allow_unreachable: true
  multimodal: Living Room Camera

camera:
  - platform: amcrest
    host: amcrest_IP_address
    username: YOUR_USERNAME
    password: YOUR_PASSWORD
  - platform: mjpeg
    mjpeg_url: IP_address/videostream.cgi?user=YOUR_USERNAME&pwd=YOUR_PASSWORD
    name: Living Room Camera
```

b) Custom settings

▲Figure 12. Configuration file example code for Sofie.



▲Figure 13. Latency in turning on light.

tional and smart home scenarios. When a user walks to a physical switch position to turn on the light (Fig. 13a), we assume T_1 is the time that takes a user to physically walk to the light switch position, and T_2 is the time to switch the light from off to on position, then the total response time is $T_1+T_2 \approx T_1$ since T_2 is perceived as instantaneous ($T_2 \approx 0$). Therefore, the latency is T_1 in this scenario. On the other hand, in Sofie the response time for turning on the light consists of three parts (Fig. 13b): The time for user sending the command by voice or user interface (T_1), that for Sofie parsing the command to determine the object and action (T_2), and that for Sofie turning on the light according to the command (T_3). For the user, the expected response time is counted after he actuates the command ($T_1 \approx 0$). Thus, the response time is T_2+T_3. As discussed earlier, the response time should be limited to 1 s for the user to interact freely and easily with the system. That is $T_2+T_3 \leqslant 1$ s. During this process, many factors could potentially contribute to the latency, including software components, hardware configuration, and network conditions. In our preliminary experiment, turning on the light via a user interface takes less than 200 ms for an event from initiation to completion in a stable Wi-Fi environment. However, if the light is turned on by using a voice-controlled service, Sofie will face more challenges by spending more time in capturing and parsing voice commands [37], [38]. This is mainly due to dependence on NLP models. Thus, a well-trained NLP model and search engine are urgently required for a satisfactory response time in Sofie.

6.2 Lessons and Experience

Whether it is a technical or non-technical task, there is a minimum skill level required to complete the task on hand. Depending on the task complexity and the field the task belongs to, knowledge in more than one field or area might be required.

Converting a non-smart home to a smart one requires some knowledge in software, hardware, and networking. For a non-technical homeowner, the Do-It-Yourself (DIY) task of converting a non-smart home to a smart one can be difficult and overwhelming. It will take several trials and errors before such a task is completed successfully. As an example, a DIY wireless surveillance system project will require 1) hardware configuration, 2) software installation and configuration, and 3) maintenance. Therefore, there is a considerable amount of manual work involved that most homeowners will find cumbersome.

Open source home automation software like Home Assistant provides a solution for integrating various home automation systems into one single solution via a uniform user interface. Although such open source software is vendor- neutral, hardware/protocol-agnostic, extensible, and platform independent, a homeowner has to invest a considerable amount of time in learning its concept and architecture in order to setup a customized smart home. The steep learning curve revolves around the setup and configuration of the system. That is, the existing open source software is neither plug and play nor a commercial off-the-shelf (COTS) product.

In addition to that, the current open source software is still not a data-oriented system; therefore, it does not involve machine learning. To further elaborate on that point, an experienced smart home user can utilize external software by creating scripts to communicate with the open source software. As an example, a user can use software like Blueiris to consume open source software API in order to trigger events. As an example, Blueiris can communicate with Home Assistant to turn on lights, TV, and so on when motion is detected. That is, any device (thing) on the home automation bus can be controlled remotely when motion is detected.

7 Conclusions

In this paper, we proposed Sofie, a smart operating system for the IoE, and discussed its design and implementation. We illustrated the design of Sofie at home and the significance of latency in a smart home environment. We then compared open source and commercial smart home systems that are available on the market nowadays. When explaining the implementation of Sofie in detail, we attempted to discuss a few of IoE-associated challenges that are related to configuration and maintenance. In general, whether it is a smart home or a connected vehicle, the user experience is very critical to the success of IoE applications. If it is hard to configure, maintain, and communicate with devices, the smart home experience will not be successful. If it takes 30 seconds to turn on a light remotely, the chances are that the user would not use the application for long. We showed that Sofie could be helpful in the IoE to practitioners to better manage their things, data, and services.

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Jun. 2016. doi: 10.1109/JIOT.2016.2579198.
[2] Cisco, "Cisco global cloud index: forecast and methodology, 2014–2019," Cisco, white paper, 2014.
[3] D. Evans, "The internet of things: how the next evolution of the internet is changing everything," CISCO white paper, vol. 1, p. 14, 2011.

[4] Y. Strengers. (2016, Jun. 10). *Creating pleasance: new needs for the smart home* [Online]. Available: http://www.demand.ac.uk/10/06/2016/creating-pleasance-new-needs-for-the-smart-home-yolande-strengers
[5] D.-L. Wang, "The internet of things the design and implementation of smart home control system," in *IEEE International Conference on Robots & Intelligent System (ICRIS)*, Zhangjiajie, China, Dec. 2016, pp. 449–452. doi: 10.1109/ICRIS.2016.95.
[6] U. Bakar, H. Ghayvat, S. Hasanm, and S. Mukhopadhyay, "Activity and anomaly detection in smart home: a survey," in *Next Generation Sensors and Systems*. Berlin/Heidelberg, Germany: Springer, 2016, pp. 191–220.
[7] Y. Strengers, "Envisioning the smart home: reimagining a smart energy future1," in *Digital Materialities: Design and Anthropology*, S. Pink, E. Ardevol, and D. Lanzeni ed. London, UK: Bloomsbury Publishing, 2016.
[8] E. Ahmed, I. Yaqoob, A. Gani, M. Imran, and M. Guizani, "Internet-of-things-based smart environments: state of the art, taxonomy, and open research challenges," *IEEE Wireless Communications*, vol. 23, no. 5, pp. 10–16, Nov. 2016. doi: 10.1109/MWC.2016.7721736.
[9] N. Jiang, C. Schmidt, V. Matossian, and M. Parashar, "Enabling applications in sensor-based pervasive environments," in *Proc. 1st Workshop on Broadband Advanced Sensor Networks (BaseNets)*, San Jose, USA, 2004, p. 48.
[10] M. Gowda, A. Dhekne, S. Shen, et al., "Bringing iot to sports analytics," in *14th USENIX Symposium on Networked Systems Design and Implementation*, Boston, USA, 2017, pp. 498–513.
[11] D. Vasisht, Z. Kapetanovic, J. Won, et al., "Farmbeats: An iot platform for data-driven agriculture," in *14th USENIX Symposium on Networked Systems Design and Implementation*, Boston, USA, 2017, pp. 514–529.
[12] E. Soltanaghaei and K. Whitehouse, "Walksense: Classifying home occupancy states using walkway sensing," in *Proc. 3rd ACM International Conference on Systems for Energy-Efficient Built Environments*, Stanford, USA, 2016, pp. 167–176.
[13] Y. Agarwal, B. Balaji, R. Gupta, et al., "Occupancy-driven energy management for smart building automation," in *Proc. 2nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building*, Zurich, Switzerland, 2010, pp. 1–6. doi: 10.1145/1878431.1878433.
[14] D. Austin, Z. T. Beattie, T. Riley, et al., "Unobtrusive classification of sleep and wakefulness using load cells under the bed," in *Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, San Diego, USA, 2012, pp. 5254–5257. doi: 10.1109/EMBC.2012.6347179.
[15] G. Gao and K. Whitehouse, "The self-programming thermostat: optimizing setback schedules based on home occupancy patterns," in *Proc. First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, Berkeley, California, 2009, pp. 67–72. doi: 10.1145/1810279.1810294.
[16] G. Zhang and M. Parashar, "Context-aware dynamic access control for pervasive applications," in *Pro. Communication Networks and Distributed Systems Modeling and Simulation Conference*, San Diego, USA, 2004, pp. 21–30.
[17] Amazon. (2017, Jun. 2). Amazon echo [Online]. Available: https://www.amazon.com/Amazon-Echo-Bluetooth-Speaker-with-WiFi-Alexa/dp/B00X4WHP5E
[18] Google. (2017, Jun. 2). Google home [Online]. Available: https://madeby.google.com/home
[19] Apple. (2017, Jun. 6). Apple homepod [Online]. Available: https://www.apple.com/homepod
[20] M. Chandak and R. Dharaskar. (2010, Apr.). Natural language processing based context sensitive, content specific architecture & its speech based implementation for smart home applications. *International Journal of Smart Home* [Online]. *4(2)*. Available: http://www.sersc.org/journals/IJSH/vol4_no2_2010/1.pdf
[21] MisterHouse. (2017, May 10). MisterHouse—it knows kung-fu [Online]. Available: http://misterhouse.sourceforge.net
[22] Home Assistant. (2017, May 10). Home assistant, an open-source home automation platform running on python 3 [Online]. Available: https://home-assistant.io
[23] OpenHAB. (2017, May 10). OpenHAB, a vendor and technology agnostic open source automation sofware for your home [Online]. Available: https://www.openhab.org
[24] HomeGenie. (2017, May 10). HomeGenie, the open source, programmable, home automation server for smart connected devices and applications [Online]. Available: http://www.homegenie.it
[25] Domoticz. (2017, May 10). Domoticz, control at your finger tips [Online]. Available: http://www.domoticz.com
[26] Freedomotic. (2017, May 10). Freedomotic, open IoT framework [Online]. Available: http://www.freedomotic.com
[27] YAML. (2017, May 10). Yaml ain't markup language [Online]. Available: http://yaml.org
[28] M. Sezgin *et al.*, "Survey over image thresholding techniques and quantitative performance evaluation," *Journal of Electronic imaging*, vol. 13, no. 1, pp. 146–168, Jan. 2004.
[29] Safehome. (2017, Jun. 2). Best home security company response times [Online]. Available: https://www.safehome.org/security-systems/best/response-times
[30] K. M. Tsui and S.-C. Chan, "Demand response optimization for smart home scheduling under real-time pricing," *IEEE Transactions on Smart Grid*, vol. 3, no. 4, pp. 1812–1821, Dec. 2012. doi: 10.1109/TSG.2012.2218835.
[31] F. Fernandes, H. Morais, Z. Vale, and C. Ramos, "Dynamic load management in a smart home to participate in demand response events," *Energy and Buildings*, vol. 82, pp. 592–606, Oct. 2014. doi: 10.1016/j.enbuild.2014.07.067.
[32] T.-Y. Chung, I. Mashal, O. Alsaryrah, et al., "Design and implementation of light-weight smart home gateway for social web of things," in *IEEE Sixth International Conf on Ubiquitous and Future Networks (ICUFN)*, Shanghai, China, Jul. 2014, pp. 425–430. doi: 10.1109/ICUFN.2014.6876827.
[33] M. Li and H.-J. Lin, "Design and implementation of smart home control systems based on wireless sensor networks and power line communications," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 7, pp. 4430–4442, Jul. 2015. doi: 10.1109/TIE.2014.2379586.
[34] Y. Ozturk, D. Senthilkumar, S. Kumar, and G. Lee, "An intelligent home energy management system to improve demand response," *IEEE Transactions on Smart Grid*, vol. 4, no. 2, pp. 694–701, Jun. 2013. doi: 10.1109/TSG.2012.2235088.
[35] R. Deng, Z. Yang, F. Hou, M.-Y. Chow, and J. Chen, "Distributed real-time demand response in multiseller–multibuyer smart distribution grid," *IEEE Transactions on Power Systems*, vol. 30, no. 5, pp. 2364–2374, Sept. 2015. doi: 10.1109/TPWRS.2014.2359457.
[36] R. B. Miller, "Response time in man-computer conversational transactions," in *ACM AFIPS' 68*, San Francisco, USA, Dec.1968, pp. 267–277. doi: 10.1145/1476589.1476628.
[37] P. Chahuara, F. Portet, and M. Vacher, "Context-aware decision making under uncertainty for voice-based control of smart home," *Expert Systems with Applications*, vol. 75, pp. 63–79, Jun. 2017. doi: 10.1016/j.eswa.2017.01.014.
[38] M. R. Abid, E. M. Petriu, and E. Amjadian, "Dynamic sign language recognition for smart home interactive application using stochastic linear formal grammar," *IEEE Transactions on Instrumentation and Measurement*, vol. 64, no. 3, pp. 596–605, Sept. 2015. doi: 10.1109/TIM.2014.2351331.

Manuscript received: 2017-06-07

Biographies

CAO Jie (jiecao@wayne.edu) received his B.S. in telecommunication engineering from Xidian University, China and M.S. in computer science from Wayne State University, USA. He is currently pursuing his Ph.D. in computer science at Wayne State University and internship at Interdigital Inc. His research interests include edge computing, computer systems, and wireless health. He has published 5 research papers and his publication of MyPalmVein received the Best Student Paper Award from HealthCom, 2015.

XU Lanyu (xu.lanyu@wayne.edu) received her B.S. in electronic and information engineering from Tongji University, China. She is currently a Ph.D. candidate in computer science at Wayne State University, USA. Her research interests include edge computing, computer systems, and cognitive service.

Raef Abdallah (raef.abdallah@gmail.com) received his B.S. in computer science from Lebanese American University, Lebanon. He holds M.S. degrees in computer science and industrial engineering from Oklahoma State University, USA. His research interests include IoT, smart homes, simulation, and design of algorithms. He has developed solutions for major companies in the United States in the areas of education, manufacturing, and defense. He is currently working in the connected vehicle technology.

SHI Weisong (weisong@wayne.edu) is a Charles H. Gershenson Distinguished Faculty Fellow and a professor of Computer Science at Wayne State University, USA. His research interests include edge computing, computer systems, energy-efficiency, and wireless health. He received his B.S. from Xidian University, China in 1995, and Ph.D. from Chinese Academy of Sciences, China in 2000, both in computer engineering. He is a recipient of National Outstanding Ph.D. Dissertation Award of China and the NSF CAREER award. He is an IEEE Fellow and an ACM Distinguished Scientist.