博　士　論　文

The Research of IoT Architecture for
Open Services in Smart Buildings

(スマートビルにおけるオープンサービスの
ためのIoTアーキテクチャーの研究)

シャフリル　バンダラ

ABSTRACT

In recent years, massive advancement in information and communication technologies have led to a new paradigm, The Internet of Things (IoT), in which every device are connected to the Internet, provide great benefits to control and monitor the physical world in real time. The deployment of IoT technologies in buildings becomes a solution to realize smart buildings. The realization of smart buildings is expected to reduce energy consumption and provide comfortable space for occupants. Currently, building systems are managed closely and can only be managed in one direction from administrator to occupants. The concept of open services has been introduced as the next paradigm in delivering services in the era of IoT. Open services have been successful in mobile phone domain to utilize various sensors and functionalities of mobile phones to provide diverse services to users. By adopting the concept of open services into smart buildings, we can encourage the third developers to bring their innovation and create new services to occupants.

However, adopting open services into smart building area is challenging because of the natural differences between the environment of smartphones and smart buildings. In a smartphone, the hardware is managed by one platform such as iOS, android or windows phone. In contrast, a smart building consists of heterogeneous devices that may use different platforms. Since a smartphone is managed by one platform, it has a standard Application Program Interface (API) to access and utilize their functionalities, whereas a smart building has a variety of devices produced by the different manufacturer without a standard API. In addition, a smartphone contains fixed elements, which mean no elements will be added or removed after the smartphone is manufactured. By contrast, a smart building contains dynamic devices, which mean devices may be dynamically attached and removed at any time according to necessity. Moreover, how to restrict an access to elements of a smartphone is managed by operating systems, while how to implement access control in buildings is challenging because it may consist of heterogeneous devices including those with limited resources. Furthermore, a smartphone is used by one

user, while a smart building is occupied by many users. Therefore, services in smart buildings are more complex compared to smartphones.

In this thesis, we propose architecture to provide open services in smart buildings by utilizing IoT devices. To deal with the heterogeneous devices, we propose a de-facto standard for API design, which adopts device abstraction and uID architecture. We also provide functionality description to allow developers create application more easily. To control access to heterogeneous devices including resource-constrained device, we propose an access control framework in which security manager is deployed as a trusted third party, and split into authentication manager and access control manager. Lastly, we investigate what services are needed in buildings and show how to realize an idea into a service. As a concrete example, we show the development of service for automatically predicting collective user preference in smart buildings.

To evaluate the proposed architecture, we implemented Smart Building API with access control framework in the real building environment and developed a service for predicting collective user preferences. A de-facto standard API design is evaluated by comparing it with the development process of application by using traditional RESTful style. The results showed that the proposed API design succeeds in reducing the development cost and supporting the developers to create application more easily. An access control framework is evaluated by conducting several scenarios of the experiment to measure the performance. The average of response time is less than 0.1 seconds, which indicates that the proposed framework is feasible to be practically used in a real smart building. Lastly, a service for predicting collective user preference is evaluated by conducting the experiment in a real smart building. The result showed that the proposed services reduced a manual intervention by 64.2 %, which indicates a service is feasible to optimize comfort level in smart buildings.

Acknowledgements

First, I would like to express my sincere gratitude to my supervisor Professor Ken Sakamura and Professor Noboru Koshizuka for their support, motivation, and immense knowledge. Their guidance helped me in all the time of research and writing of this thesis.

I am also very thankful to Dr. Takeshi Yashiro from YRP Ubiquitous Networking Laboratory for helping me throughout my research and giving continuous encouragement.

I would like to thank my wife, Yuri and my family in Indonesia and Shizuoka, for their love and support. My family have been important for their help and encouragement, and for making every-day life so easy and enjoyable.

I would like to offer warm thanks to all staff and member of Sakamura-Koshizuka Laboratory for their support by providing such a great working environment.

Finally, I would like to thank Sato Yo International Scholarship Foundation for their finance support during my study as a Ph.D. student at the University of Tokyo. Thank you!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In recent years, massive advancement in information and communication technologies have led to a new paradigm, The Internet of Things (IoT) [1], in which a variety of devices in the physical world are connected to the Internet. The number of connected devices is predicted to grow rapidly over the next decade. Cisco IBSG predicted there will be 50 billion devices connected to the Internet by 2020 [2]. Similarly, according to Gartner, Inc., there will be 20 billion connected things by 2020 [3], and the estimation of ABI Research shows that there are more than 30 billion wirelessly connected devices in the market by 2020 [4].

The growing number of IoT devices give a great impact on every aspect of our daily lives. By connecting devices into the Internet, those devices are able to interact with each other and cooperate with other things to compose new services. Furthermore, IoT devices which have sensing capabilities can be utilized to monitor the change of circumstances in the physical world in real time. Real-time monitoring allows systems to take an action for every change, and enable the concept of automation which makes our lives easier, more convenient, and more comfortable.

IoT devices can be utilized in a various domain, including healthcare, manufacturing, farming, transportation, education, energy, government, and building. In healthcare domain, IoT technologies can be exploited for remote tracking and

monitoring behavior of patient [5], remote treatment and surgery, and etc. In manufacturing domain, it can be utilized for equipment monitoring and management [6]. In the farming domain, it can be applied to monitor environmental conditions remotely [7]. In transportation domain, it can be exploited to build traffic signals that respond to traffic conditions [8]. In education domain, it can be utilized for remote education and engagement [9]. In energy domain, it can be applied for smart metering and time-of-use billing for consumers [10]. In government domain, it can be used to realize smart cities and smart communities [11]. In the building domain, it can be exploited for building automation [12].

The building is one of the domains which is got attention in the research community. The deployment of IoT technologies in buildings is believed to be a solution to realize smart buildings. The realization of smart buildings is expected to reduce energy consumption and provide comfortable space to occupants. Nowadays, buildings are estimated to account for about 40% of the global energy consumption and contribute over 30% of the CO_2 emissions [13]. Moreover, the environmental comfort of a building is strongly related to the occupant's productivity and quality of life and work because people are expected to spend more than 80% of their lives in buildings [14]. Before the emergence of IoT concept, those problems has been addressed in research of intelligent building. However, the previous work mainly focused on how to develop intelligent building technology and performance evaluation methodologies [15]. In addition, though the intelligent building has emerged, the management is still closed and one direction from administrator to occupants. Therefore, the number of services can be provided to occupants are still limited.

The concept of open services has been introduced as the next paradigm in delivering services in the era of IoT. Open services have been successful in mobile phone domain to utilize various sensors and functionalities in a smartphone and provide diverse services to users. Contrast to the traditional mobile phone which applications are provided by the manufacturer, the platform in a smartphone is open to the third developers. The information how to utilize a various of sensors and functionality is published. Therefore, the third developers are able to involve

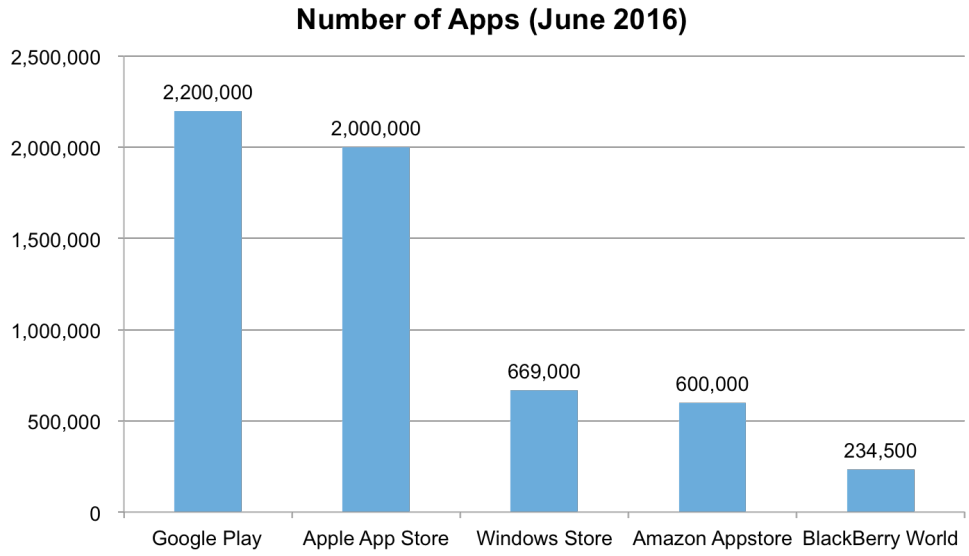


Figure 1.1: Number of Smartphone Applications

their innovative idea to combine heterogeneous sensors and functionality to provide new services. As a result, currently, users of a smartphone are flooded by a plethora of applications which can make their activities more convenient. The Fig. 1.1 shows the number of applications available in several applications stores [16]. By adopting the concept of open services into smart buildings, we can encourage the third developers to bring their innovation and create new services to occupants.

However, adopting open services into smart building area is challenging because of the natural differences between the environment of smartphones and buildings. The Fig. 1.2 shows the comparison between smartphone and smart buildings environment. In a smartphone, the hardware is managed by one platform such as iOS or android. Conversely, a smart building consists of heterogeneous devices that may use different platform each other. Since a smartphone is managed by one platform, it has a standard API (Application Program Interface) for access and utilizes their functionality, whereas a smart building which has a variety of devices that produced by different manufacturers, does not have a standard API. In addition, a smartphone contains fixed elements which mean no elements will

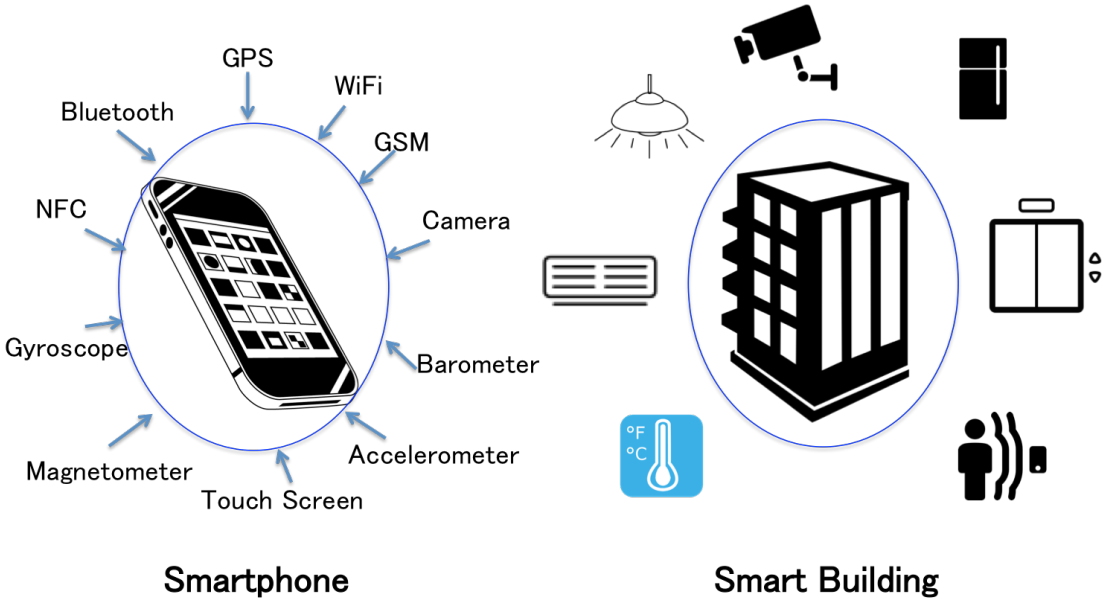


Figure 1.2: Comparison of Smartphone and Smart Building

be added or removed after the smartphone is manufactured. By contrast, a smart building contains dynamic devices which mean devices may be dynamically attached and removed at any time according to necessity. Moreover, how to restrict an access to elements of a smartphone is managed by operating systems, while how to implement access control in buildings is challenging because it may consist of heterogeneous devices including those with limited resources. Furthermore, a smartphone is used by one user, while a smart building is occupied by many users. Therefore, services in smart buildings more complex compare to smartphones.

In this work, we focus on how to utilize IoT devices for providing open services in smart buildings. Firstly, how to standardize service of heterogeneous IoT devices in smart buildings is addressed. Secondly, we describe how to to minimize manual procedure on developing applications by extending RESTful API style. Thirdly, we propose a framework to handle access control for a diversity of devices in smart buildings. Finally, we investigate services needed in smart building and implement one concrete example of service.

1.2 Research Issue

The deployment of IoT devices in buildings has a great potential for realizing smart buildings. By utilizing and combining those devices, a variety of services can be delivered to occupants. To allow the third developers to create innovative services, the adoption of open services is needed. The administrator of buildings publishes device APIs on the developer site. The third developers bring their innovative idea and reference the developer site when creating new applications. New applications deliver a service to occupants. To realize this scenario, in this thesis we address some of the research issues as below:

- how to standardize service of heterogeneous IoT devices
- how to minimize manual procedure on developing applications
- how to control access to a diversity IoT devices
- how to support a realization of ideas into services

These issues are important to realize open services in the smart building environment. In order to solve these issues, we present our comprehensive study. We have focused our contributions, listed in the next section, to address these issues. In the next subsection, the explanation of these issues is presented.

1.2.1 How To Standardize Service of Heterogeneous IoT Devices

In order to maximize the utilization of IoT devices, interoperability among a plethora of connected devices becomes a crucial aspect. To increase the interoperability, such devices are often equipped with RESTful API [17], in which how to communicate and exchange data with devices are defined. By opening the device API to the public, developers can bring their innovation and develop new applications to provide service to occupants. In order to let developers create applications more easily, it is necessary to standardize the device API. However, standardization

of device API is considered challenging, because it may consist of heterogeneous devices with providing different services as each device has different specification and configuration of API. Currently, each manufacturer defines specification and design API for their device products separately. Without the standardization of API, it is difficult to realize physical mashups and combine IoT devices freely.

1.2.2 How To Minimize Manual Procedure on Developing Applications

The emergence of innovative applications depends on application development cost. In application development process, the third developers reference APIs of devices to be used which is published in developer site. Due to the simplicity, the RESTful style widely used for describing how to communicate and exchange data with devices. However, the traditional RESTful API, which is designed based on a resource-oriented paradigm, lacks in functionality description of resources [18]. As a result, to create new applications which utilize IoT devices, the developer must understand the specification of all device APIs that they want to use, which leads to a high number of manual procedure and increase development cost.

1.2.3 How To Control Access To A Diversity IoT Devices

The security cannot be disregarded from the employment of IoT devices because the impact of the security breach in these devices tends to be larger, as it directly affects the physical environment we live in. If an unauthorized user is able to access IoT devices in smart buildings, it may harm an occupant's life for instance by locking a door, turning off lights or air conditioners, which make occupants lose their comfortable spaces. However, the conventional access control framework from the cyber world cannot simply be adopted by these devices because it may consist of heterogeneous devices including those with limited resources. Furthermore, appliances and devices in smart building environment may be dynamically attached and removed at any time according to necessity. Therefore, it is crucial to investigate an access control framework by taking into account the specific characteristics of

the physical world in smart environments.

1.2.4 How To Support A Realization Of Ideas Into Services

Compared to smartphones, services in smart buildings are complex due to the number of users and a diversity of devices. It is easy to developers of smartphone applications to imagine the use of smartphone because a smartphone is used by a user. In addition, elements and functionalities of a smartphone are fixed when it manufactured so that it is clear what materials to use in applications to be developed. In contrast, a smart building is occupied by many users which are different user preference. Moreover, the number of devices in smart buildings may be dynamically changed due to necessity. As a result, it is difficult to the developer to prepare for developing new applications. In order to realize open services in smart buildings, it is crucial to investigate what services are needed in buildings and show how the ideas of services can be realized by combining various IoT devices.

1.3 Contribution

In this work, we focus on the above research issues, and the main contribution of this thesis are:

- Firstly, a de-facto standard of API design for open services in smart buildings is presented. In order to standardize device API, we need device abstraction, in which a diversity of devices in IoT environment can be simplified. The abstraction of devices is made by investigating the property of devices. As a result of our investigation, we found that devices have two properties: the static property named as *attribute*, and the dynamic property named as *state*. In contrast to traditional RESTful, which all of the properties are designed to be the endpoint of URI, we design each device API only has two endpoints of URI: *attribute* and *state*. By adopting this abstraction, we can realize a de-facto standard of device API and overcome the diversity of devices in IoT

environment. To handle the lack of functionality description, we explore the useful information for describing devices in the development process, such as parameters can be set and the range of parameters. Such information can be used to generate a menu for controlling device automatically. We included that useful information in the static property, *attribute*. To provide services to occupants, we also need context data. Therefore, we also present context APIs, which is designed based on observation of relationship from three entities in smart buildings: *space*, *occupant*, and *device*. Device APIs and context APIs are published at the developer site, which can be referred when developing new applications.

- Secondly, an access control framework for API-enabled devices in smart buildings is proposed. To accommodate the dynamic environment in smart building the security manager is deployed as a trusted third party in the system. Complex security computations are delegated to the security manager so that resource-constrained device can be securely handled. To support scalability and efficiency the security manager is split into authentication manager and access control manager. Authentication manager issues an access token for the verified user. To support a various authentication method, which has different levels of certainty, the idea of confidence level is introduced. After a device received a request from users, the device is forced to ask access control manager to evaluate the received request. The access control manager evaluate the request based on the access control policies, which are composed by the administrator in advance.

- Thirdly, how to realize an idea into a service is shown. First, we investigate what services are needed in buildings, and classify it. There is two main type of services, service for energy efficiency and comfortable environment. Since a space on a building is shared by multiple users which have their own preference, it is challenging to provide comfort for multiple users. Then, we propose an algorithm for automatically predicting collective user preference in smart

buildings. The building spaces are occupied by a group of occupants who belong to an organization. In an organization, usually, some kind of hierarchy exists between the occupants. We observe how this organizational hierarchy influences a decision of comfort conditions for multiple users. When a group of occupants with different authority gathers in one place, the occupant who has the greater authority tends to decide the comfort conditions. From this observation, we design an algorithm to predict user preference for a group of occupants. We define a model for a group of occupants who share building space. A relationship between the occupants can be defined by investigating in the group who has changed the current conditions in order to satisfy his own comfort conditions. The comfort conditions for the group can be obtained from the last applied conditions. The relationship of the occupants and the comfort conditions are generated by monitoring the interaction between the occupants and the environment. The collective user preference can be predicted from the relationship of the occupants that have been defined and the comfort conditions that have been obtained. Based on the prediction, the system is able to provide an optimal comfort level for the occupants in smart buildings.

1.4 Outline

The remainder of this thesis is organized as follows.

- In chapter 2, we summarize prior work related to IoT architecture, security framework in a smart environment and services in smart buildings. Firstly, we present existing platforms that proposed to deal with the complex of IoT environment. Secondly, we discussed previous works have been proposed to enforce security in smart environments. Thirdly, we explore existing research related to services in smart buildings.

- In chapter 3, we present a de-facto standard API design, which is part of IoT

architecture for open services in smart buildings. We start this chapter by describing the flow of application development in open services environment. Then, we propose an IoT architecture to allow developers create application more easily. We also describe in details how we implemented the proposed architecture. Then, we describe how to evaluate the proposed architecture.

- In chapter 4, we describe an access control framework for open services in smart buildings. First, we explain the concept of open services in our work. Second, we present design of access control framework for securing open services environment. After describing the implementation of the proposed framework, we discuss how we evaluate the access control framework.

- In chapter 5, firstly, we investigate what services are needed in buildings, and classify it. Secondly, we explore how to optimize comfort for occupants, which have their own preference and sharing spaces each other in buildings. To overcome this condition, we propose an algorithm for automatically predicting collective user preferences. The prediction is conducted by observing the relationship between occupants and the last condition applied to the group of occupants. Thirdly, we describe how to implement and evaluate the proposed algorithm.

- In chapter 6, we discussed how the proposal in this thesis addresses the issues listed in chapter 1. We also combine the proposal in the previous chapter as an IoT architecture to utilize heterogeneous devices for realizing open services in smart buildings.

- In chapter 7, we conclude this thesis and discuss directions for future research.

Chapter 2

Related Work

In this chapter, related works are presented. Firstly, prior works related to IoT architecture are discussed. Secondly, the previous works on security framework in the smart environment are described. Thirdly, the prior works related to service in smart buildings are introduced.

2.1 IoT Architecture

The phrase "Internet of Things" (IoT) first used by Kevin Ashton of MIT Auto-ID Center in 1999 which refers to a paradigm shift from the current Internet that almost depends on human-entered data to the computer that able to observe, identify and understand the physical world without any support from human beings [19]. Actually, the similar concept of integrating computers seamlessly into the world, "Ubiquitous Computing", was coined by Mark Weiser in 1999 [20]. Previously, the similar concept of combining computer systems in a loosely-coupled manner for providing services, " Highly Functionally Distributed System" (HFDS), has been proposed as the final goal of the TRON Project which started in 1984 by Ken Sakamura [21]. These terminologies have the same vision to distribute computers to the physical world and integrate it into the system to provide services.

In the last few decades, IoT has become an attractive research area. Much work has been done to address IoT-enabling technologies including identification, sens-

ing, and communication technologies such as RFID [22], NFC [23], Wireless Sensor Networks [24], and etc. After things can be identified, how to uniquely identify it become a crucial issue. To overcome this problem, Koshizuka et al. proposed a ubiquitous ID (uID) architecture, which adopted ubiquitous code (ucode), 128-bit unique identifier, and ucode Relation (ucR) Model, a standard framework for knowledge-level context description[25].

After the physical objects can be uniquely identified, they must be able to communicate with the Internet. The effort to connect physical objects have been done since the early of the 90s. The first Internet-connected device was created by John Romkey in 1990 [26]. He developed a toaster that could be turned on and off over the Internet. He worked together with his colleague, Simon Hackett, to connect the toaster to the Internet with TCP/IP networking, and used a Simple Networking Management Protocol Management Information Base (SNMPMIB) to control it. In 2000, LG introduced the first connected refrigerator, which has a webcam that is used to scan what is inside the refrigerator. In 2011, Google presented Smart Nest Thermostat, a smart device for controlling a central air conditioning unit based on heuristic and learned behavior [27].

Many standards organization including World Wide Web Consortium (W3C), Internet Engineering Task Force (IETF), EPCglobal, Institute of Electrical and Electronics Engineers (IEEE), and the European Telecommunications Standards Institute (ETSI) have proposed a plethora of protocols in support of the IoT. The protocols can be classified as infrastructure protocols, service discovery protocols and applications protocols [28]. As infrastructure protocols, protocol stacks based on IEEE 802.15.4, such as WirelessHART [29], Zigbee [30], have been proposed to support low power communication. In order to enable IP connectivity in resource constrained networks over IEEE 802.15.4, the IPv6 over Low-Power WPAN (6LowPAN) [31] have been proposed. As service discovery protocols, multicast DNS (mDNS) [32] and DNS Service Discovery (DNS-SD) [33] have been introduced for discovering resources and services offered by IoT devices. In application protocols, Constrained Application Protocol (CoAP) [34], Message Queue Telemetry Trans-

port (MQTT) [35] [36] and Extensible Messaging and Presence Protocol (XMPP) [37] have been proposed. CoAP was designed by IETF to provide Representational State Transfer (REST) architecture for constrained devices. MQTT was released by IBM to provide lightweight M2M communications. XMPP was standardized by IETF to provide a real-time exchange data between any two or more network entities.

Since the number of Internet-connected devices is expected to grow rapidly and reach 50 billion in by 2020 [38], the next challenge is how to utilize and combine heterogeneous IoT devices. Much work has been done for addressing a diverse, complex, and heterogeneous environment of IoT. To integrate real-world devices to the Web, Guinard et al. discussed how the REST principles can be applied to embedded devices [39]. They proposed two ways to integrate real-world devices to the Web: implementing Web server on tiny embedded devices and deploying intermediate gateway that offers a unified REST API. They have also shown the adoption of REST API to real-world devices provides great benefit to prototype applications in a flexible mechanism. Since the REST API has a limitation in describing services, Panziera et al. proposed a framework for self-descriptive RESTful services [18]. They proposed five best practices for representing characteristics of a service. Then, they introduced a software framework for semi-automatically extracting service information from provider documentation. To allow the integration of constrained devices, IETF The Constrained RESTful Environments (CoRE) working group have introduced discovery and self-description mechanism based on a web link format suitable for constrained devices in RFC 6690 [40].

Opening IoT infrastructure to the third party, or deliver open services also becomes an active research area. To establish an ecosystem for IoT, Kim et al. introduced an open IoT service framework [41]. They defined stakeholders of IoT ecosystem and proposed a framework to facilitate stakeholders to join IoT ecosystem. They implemented their framework and discussed iBike application as a use case. They also showed that OpenIoT platform brought convenience for stakeholders. In [42], to allow developers to build more innovative applications through open

APIs, Elmangoush et al. introduced a standard APIs for M2M communications including Network APIs, Device APIs, and Data APIs. In order to coordinate and integrate data and services, Charalabidis et al. proposed the Gov4All governance framework, a framework for supporting open application development in [43]. They implemented a prototype of the proposed framework on four major pillars: applications, services, open data and community, where each pillar provided a friendly user interface. In [44], Serrano et al. discussed how to support data interoperability in a design of the Future Internet by adopting semantic technologies, open service frameworks and information models. They also introduced how to utilize the concept of Linked Data for supporting information interoperability in service composition and management processes. In order to integrate the existing isolated vertical system, Mingozzi et al. proposed Building the Environment for the Things as a Services (BETaaS) framework in [45]. BETaaS framework consists of four layers: service layer, thing as a service (TaaS) layer, adaption layer and physical layer, where a set of APIs are defined as an interface between the layers. In order to support easy integration of existing models and guarantee reusability, the proposed architecture is kept to be opened.

To deal with the complex IoT environment, some platform have been proposed in [46], [47], [48] and [49]. In [46], Mayer et al. investigated how to facilitate the composition of services for end users. To make end user enables to configure their environment, they proposed goal-driven approach. The users could generate their goal with a graphical editor. They also extend RESTdec [50] with the concept of states and state changes that allow using the system to describe any service that induces state transitions.

In [47], Akasiadis et al. explored how to develop complex service in IoT ecosystem. To integrate devices, services, and humans into applications, they presented IoT compatible platform, SYNAISTHISI, which services are semantically enriched using ontologies. They also implemented a smart meeting room ontology and shown how a developer can build service which determines the number of people inside a smart room.

In order to link the existing embedded systems into the IoT network, Yashiro et al. presented uID-CoAP architecture in [48]. Since the most of the existing framework designed solely for extremely resource-constrained devices, they focused on the framework for integrating regular embedded devices. They utilized uID architecture to handle semantic knowledge of IoT nodes and how to cooperate it. They adopted CoAP to provide a lightweight communication in embedded devices. They have designed the proposed framework on the top of T-Kernel [51], a real-time operating system used commonly in embedded systems. They implemented Home Energy Management System (HEMS) application on top of the framework as a case study.

In [49], Abeele et al. investigated how to improve the integration of a wide variety of constrained devices in the Internet of Things. They proposed cloud-base platform to integrate heterogeneous embedded devices by offering a uniform CoAP interface on a virtual device abstraction. They also implemented a dashboard to manage constrained devices and shown that developers can easily integrate constrained devices into third party services.

How to develop IoT technologies in building environments also becomes an interesting field. In [52], Carrillo et al. introduced an IoT framework to efficiently manage a smart building using a cloud computing. The framework consists of several components, including cloud services in Microsoft Azure, Rasberry Pi as the main gateway, and REST API to interact with the whole services. The framework has been implemented and evaluated in a real building.

Hou et al. discussed how to minimize the deployment time of applications and enabling cloud service for IoT in [53]. They proposed the design and implementation of APIs for IoT Cloud that supported both HTTP and MQTT protocols. They also conducted several experiments to evaluate the performance of the proposed APIs. However, they focused on the diversity of protocols without addressed the diversity of devices.

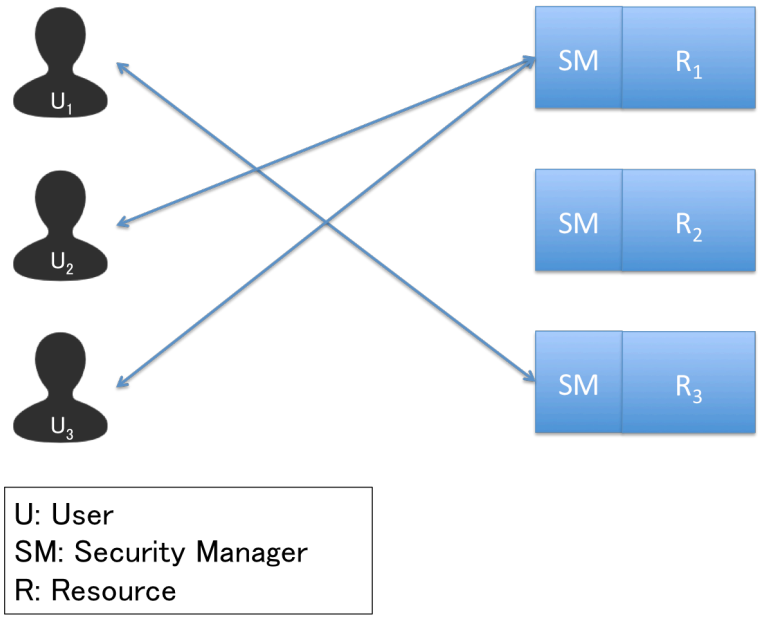


Figure 2.1: In-Situ Security Framework

2.2 Security Framework for Smart Environment

There are two kinds of conventional frameworks for enforcing security to resources: the in-situ security framework, and the proxy security framework. The Fig. 2.1 shows the architecture of in-situ security framework. The in-situ security framework is implemented by deploying a security manager in resource side. This approach has been popular in home appliances by secured them with a password or a key. A user, which tends to access device, has to attach a key or a password. Each invocation from a user is evaluated by the security manager, which is implemented in the device. However, the physical world like smart buildings consists of heterogeneous devices including resource-constrained devices. The implementation of the security manager in the resource-constrained devices is challenging because security enforcement has a complex computation that needs an adequate resource computation. The Fig. 2.2 shows the architecture of proxy security framework. The proxy security framework is implemented by placing a security manager be-

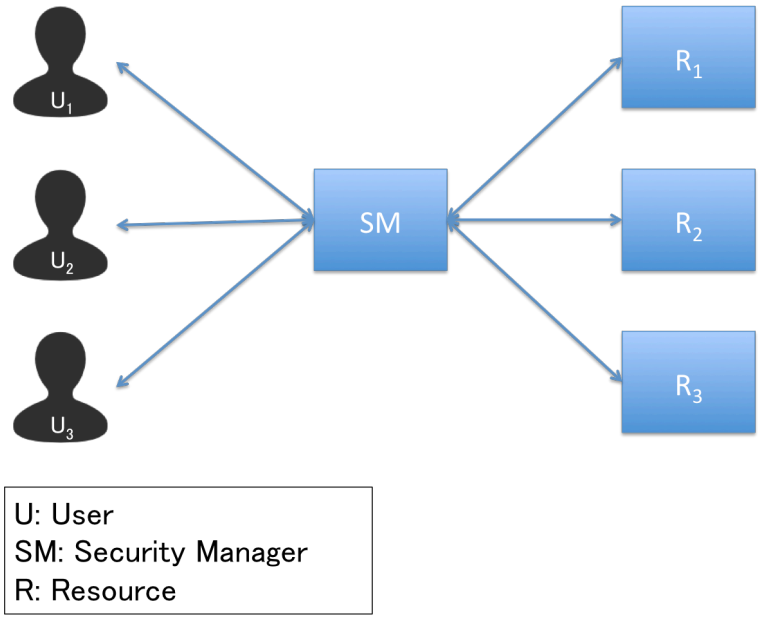


Figure 2.2: Proxy Security Framework

tween user and resource. In this approach, the security manager accepts and checks every invocation that is requested by users. This kind of framework is widely used in web technology. However, devices in smart environments may be dynamically attached or removed at any time according to necessity. With this approach, every modification on resource side has to be adapted on the security manager because it serves each invocation to resources. For example, if we deploy a new *sensorA*, in order to make *sensorA* accessible to users, we have to modify the security manager. Therefore, this approach may not appropriate for the smart environment.

Many solutions have been proposed to enforce security in smart environments. Wang et al. [54] proposed authorization models for ubiquitous computing environment adopting usage control. They contributed on extending the ongoing continuity for authorizations, obligations, and conditions. They also discussed architecture solutions for ubiquitous computing access control based on reference monitor. They analyzed two kind of reference monitor: Server-side Reference Monitor (SRM), and Client-side Reference Monitor (CRM). However, they focused on

authorization model by extending the concept of usage control and do not address how to deal with the heterogeneity of devices in the smart environment.

Kulkarni et al. [55] presented a context-aware RBAC model by extending the NIST RBAC model. The proposed model consists of two layers, context management layer, and access control layer. In the context management layer, several context agents are deployed for continuous sensing of conditions in the environment. In the access control layer, context-based permission activation is supported. They presented how the proposed model uses in two context-aware applications, patient information system and music player system. However, access control mechanism strongly relies on authentication process which is not addressed in this work.

Al-Muhtadi et al. [56] proposed an authentication framework that supports multiple authentication devices and methods. They presented assignment of different confidence values to different authentication methods. They also introduced a confidence-builder module for combining multiple confidence values. The proposed framework has been implemented in the Gaia research project. However, how to overcome the characteristics of the physical world like the heterogeneity of devices, the dynamic environment was not addressed.

Oh et al. [57] presented an access permission control mechanism that adopted resource-oriented architecture for Web-enabled things. They analyzed and described the requirement fort the access control to the resources of WoT. They proposed a decentralized mechanism, which an access control matrix is attached to every resource. They also presented how to make access control matrix scalable by changing the categorization from Subject-based to Operation-based which has four types C, R, U, and D. However, they focused on decentralized access control permission without addressing authentication process which is an important aspect of succeeding authorization mechanism.

Al-Rabiaah et al. [58] proposed a design of security framework which allows authorized applications only to know the context information. They mentioned several threats that face context information and extended Cerberus by addressing those mentioned threats. However, they focused on enforcing security between

context provider and receiver, which is insufficient to enforcing security for API-enabled devices.

Kim et al. [59] proposed OSGi (Open Service Gateway initiative) -based framework to integrate devices and services in smart home systems, including an access control model, which is realized by combining OSGi User Admin service and XACML (eXtensible Access Control Markup Language). However, they focused on supporting a variety of protocols, and the important aspect such as security, scalability, and performance of the proposed mechanism is not evaluated.

In order to secure the Web of Things (WoT), Barka et al. proposed an architecture, which adopted role-based access control (RBAC) in [60]. As the core control mechanism, they adopted the concept of a reference monitor (RM) to enforce access control policies in WoT environment. The reference monitor consists of two components: access control enforcement facility (AEF) and access decision facility (ADF). Every invocation to access an object in the system is monitored by AEF. The invocation received by AEF will be forwarded to ADF for evaluating. ADF will decide to approve the invocation or not based on access control policies.

In [61], Ho et al. discussed several defenses to mitigate the attacks in IoT devices. As a target, they selected one of important IoT devices, home smart lock system. First, they investigated the system design and properties of five popular commercial smart locks. Then, they introduced two categories of attacks on smart locks and revealed that all of the existing locks are vulnerable. To validate that user is near to the door in the verification process, they proposed touch-based intent communication (TBIC) protocol. They also discussed three possible technologies to realize their proposal: capacitive coupling, galvanic coupling, and bone conduction. They introduced *Vibrato*, a prototype of TBIC using bone conduction. Lastly, they evaluated how secure *Vibrato* by conducting several experiments.

Fysarakis et al. [62] presented Web Services Access Control for devices (WSACd), an access control framework to enable access control management of the heterogeneous embedded devices in smart home environments. By combining XACML and DPWS (Device Profile for Web Services), the interoperability among services

provided by the resource-constrained device is protected by a predefined access control policy. However, the author did not analyze the scalability of the proposed framework, which is a crucial aspect because the networked device is expected to grow. Moreover, the user authentication which is an important part of access control framework is not addressed.

Cirani et al. [63] proposed an OAuth-based authorization framework which targeting HTTP/CoAP services. Since implementing OAuth-based authorization mechanism in Service Provider is hard, they delegated the authorization functionality to the entity, denoted as OAuth Service. They evaluated energy consumption in Service Provider, which showed that they framework showed higher energy consumption due to the communication between Service Provider and OAuth Service. This work is very similar to the one presented, it focuses mainly on OAuth-based framework while the proposed one is intended to design the framework by addressing the heterogeneity of API-enabled devices and dynamic environment in smart buildings.

2.3 Service in Smart Buildings

The phrase "Intelligent Building" was coined in the United States in early of 1980s [15]. The definition of intelligent building has been evolved due to the advances in technology. In early of the 1980s, intelligent building was related to building automation as well as the definition from Cardin, which defined intelligent building as a building which has fully automated building service control system [64]. After 1985, the concept of intelligent building referred to buildings which responsive to change over time. Since 1992, the definition of intelligent building has shifted to the effective building, which can maximize the efficiency of the occupants and minimize the cost associated with the building. The phrase "smart building" emerged later due to the utilization of smart sensors, smart materials, and smart meter within buildings. According to Buckman et al., smart buildings are intelligent buildings but with additional, integrated aspects of adaptive control,

enterprise and materials and construction [65].

Up to now, many works have been proposed to deliver services in smart buildings. The services in smart buildings can be divided into two types: services for reducing energy consumption and services for increasing comfort to occupants. In order to achieve both energy saving and increasing customer satisfaction, Davidson et al. presented a multi-agent system that monitors and controls an intelligent building [66]. They defined four types of agents: personal comfort agents, room agents, environmental parameter agents and badge system agent, and programmed a number of general rules in order to achieve energy saving and occupants satisfaction.

To reduce energy consumption in buildings, Chen et al. proposed a high-level architecture for smart building control system [67]. Since the reduction by implementing individual energy efficient technologies alone would not be sufficient, they investigated how to adopt holistic, integrated approach that encourages shared responsibility among the stakeholders. They divided the system into five major domains: physical system domain, system integration domain, process integration domain, business integration domain and external input domain, which each domains containing a related set of services or functions. They implemented a prototype system and evaluated their system in a simulation.

In order to provide building control and energy saving services in the building, Byun et al. presented self-adapting intelligent system which consists of the self-adapting intelligent gateway (SIG) and sensor (SIS) [68]. Firstly, they addressed the limitations of existing context-aware systems including centralized architecture, fixed rule-based control and limited network lifetime. To overcome the listed limitations, they proposed the self-adapting intelligent system for building control and energy management services. They implemented their system in the real environment and conducted the experiments to evaluate it.

The utilization of IoT framework for realizing smart buildings have been proposed in [69] and [52]. In [69], Pan et al. presented smart location-based automated energy control IoT framework for improving energy efficiency in buildings. Since buildings are complex systems and many factors can affect the total energy con-

sumption in buildings, they monitoring the energy consumption in the real building for one year and build a model to investigate the energy consumption patterns. The proposed framework consists of five components: mobile device-based distributed energy monitoring and remote control, location application on a smartphone, multisource energy-saving policies, cloud-computing platform-based data storage and energy data modeling and strategy formation.

The study of involving user preference to achieve comfort have been proposed in [70], [71], [72] and [73]. In [70], Jazizadeh proposed a framework for ubiquitous and real-time interactions between building and human to improve thermal comfort in buildings. The occupants provide their preference by moving the slider in intermediary interface via their smartphone. The feedback from the occupants is used to control HVAC system and also used as an index of comfort perceptions. They used a fuzzy based algorithm to extract the pattern and develop a predictive model. They conducted BMS controller experiments to validate that the proposed framework was effective to improve occupant‘s comfort.

In order to automatically predict preferences based on past experience, Chen designed a context-aware recommendation system based on collaborative filtering process in [71]. The concept of Collaborative Filtering (CF) introduced to predict the impact of context on an activity by leveraging past user experiences. They have addressed two problems regarding incorporating context into CF: how to manage context in the user profile and how to measure similarities between contexts. To overcome the listed problems, they described modeling context data in CF user profiles and applied CF techniques for measuring context similarity. Lastly, they proposed an algorithmic extension to generate a prediction.

To provide user-centric comfort services while considering energy saving, Moreno et al. [72] proposed an energy-efficient management system based on Internet of Things (IoT) approach. The proposed platform based on *CityExplorer* solution [74], which have two main components: Home Automation Modules (HAM) and the supervisory control and data acquisition (SCADA). To achieve both occupant‘s comfort and energy savings, they presented Smart Comfort and Energy Control

System, which contains two modules: *Smart Comfort Prediction* and *Efficient Comfort Management*. *Smart Comfort Prediction* predicted the optimum comfort conditions according to the occupants, their activities, their location, and their individual comfort preferences. After comfort conditions have been estimated, *Efficient Comfort Management* provided the optimum comfort settings for the involved appliances that ensure minimum energy consumption. They deployed the proposed system in the real building and evaluated three aspects: identification and positioning of occupants, user-centric service prediction and energy-efficient management of the appliances.

In order to maximize occupant comfort in building, Schumann et al. [73] presented an algorithm for predicting comfort votes of occupants. They addressed the existing approach based on Predicted Mean Vote (PMV) that require many environmental data, which is costly and require precise personal dependent data, which are difficult to obtain. They introduced an algorithm, which depends only on the temperature measurement and the comfort votes. In order to identify the relevant database entries, which contains temperature, vote, and occupant, they defined a degree of similarity between two states based on distance measures. Lastly, they compared the proposed algorithm with two existing algorithm by using a collection of individual comfort votes from different countries.

Since the smart environment like a smart building is inhabited by many occupants, some previous works related to predicting multiple user preferences have been presented in [75], [76] and [77]. In [75], Yu et al. proposed building automation model which considers preferences and feedbacks from multiple users. To realize continuous reconfigurations, they introduced user model, which store preferences of individual users, and generates a final policy based on the user-based information. They design and implement the model on Multi-Agent System (MAS) architecture, which contains product agent, resource agent, order agent and staff agent, to dynamically control Building Automation System (BAS) and provide convenient for occupants. They also presented credit system for generating policies form of multiple user preferences. Each user is given 100 credits, which they may spend

on various preferences. A preference item's final setting is chosen to be that of the user who spends the most credit.

Yamamoto et al. [76] proposed a context-aware device control method for general users with different preferences on controllable targets in public smart spaces. The maximum of comfort levels was obtained by summing up comfort level functions of all users and select the maximum value and applied the value to a device. Since the public spaces are a dynamic environment, they also discussed how to efficiently collect and estimate user preference. To evaluate the proposed method, they conducted the experiment by collecting the questionnaires for 2 months from five examinees.

In order to provide suitable service to the users, Lin et al. [77] proposed a three-layer system for learning inhabitants' behaviors based on the observations from sensors. In the first layer, raw data from sensors was interpreted based on domain ontology and location information of each occupant. At the second layer, they adopted Dynamic Bayesian Networks (DBNs) for modeling temporal information. In the third layer, Bayesian Networks (BNs) was used to integrate service information, personal information and environment information to provide the adequate services to occupants. The performance of building preference model have been evaluated in the real environment.

However, the relationship among the occupant was not addressed in the previous work. In [75], credit system has been presented to solve the difference of user preference. Since multiple users in a building have a hierarchical relationship to each other, this approach may not be practical in various situations. In [76], a summation of comfort level functions of all users has been presented. Such approach may not feasible for optimizing comfort level in the smart building since the summation of the comfort level of all users not always be a collective user preference of the occupants. In [77], Multi-user Interaction Model that used Bayesian Networks is presented. The goal of the model is to infer the appropriate a *group service* for inhabitants. Their works focused on inhabitants in a smart home. Therefore, it only discussed inference a *group service* without addressing the relationship between

occupants. Since the relationship among inhabitants in a smart home is different to the relationship among occupants in the smart building, their approach may not be suitable for smart building environment.

2.4 Summary

Much works have been proposed to deal with diversity and complexity in IoT environment. However, how to design a standard API for reducing application development cost has not been addressed properly. An alternative approach to extend the traditional RESTful API, which is lacking resource description, is necessary. In this work, first, we focus on how to design a de-facto standard API of heterogeneous IoT devices, which provide various different services, for realizing open services in smart buildings.

Although much work has been done to enforce security in smart environments by integrating context information to access control model and managing access control on heterogeneous protocol devices, it is necessary to address the heterogeneity of API-enabled devices and the dynamism in smart environments. Moreover, more work is needed to implement and evaluate framework in real environments. The next focus on this work is how to develop an access control framework which deals with the heterogeneous devices and dynamic environment in smart buildings with the implementation and evaluation in real environments.

Although services in buildings become an attractive area in the research community, the previous works mainly focused on providing services rather than how to develop a service. The next focus in this work is how open API can be utilized to support a realization of ideas into services. As the concrete example, we will show how to develop a service for predicting collective user preference by utilizing open API.

Chapter 3

De-Facto Standard API Design

In this chapter, we present a de-facto standard for API design, which is part of IoT architecture for open services in smart buildings. Firstly, we describe the flow of application development in open services environment. Next, we propose an IoT architecture to allow developers create application more easily. We also describe in details how we implemented the proposed architecture. Then, we describe how to evaluate the proposed architecture. Finally, the summary is provided at the end of the chapter.

3.1 Application Development

In this section, we describe how to develop applications for open service in smart buildings and what is the problem when developing applications. The Fig. 3.1 shows the general flow of application development in smart buildings environment. How to exchange data and communicate with the IoT devices in smart buildings are described in developer site. To realize open services, developer site is disclosed to the public. Everyone can bring their innovation to utilize IoT devices and develops new applications. By referring the developer site, they can establish new applications to provide services for occupants of the building.

Let us suppose that we are going to develop a new application, named Smart Room Application, in which occupants are able to monitor and control the devices

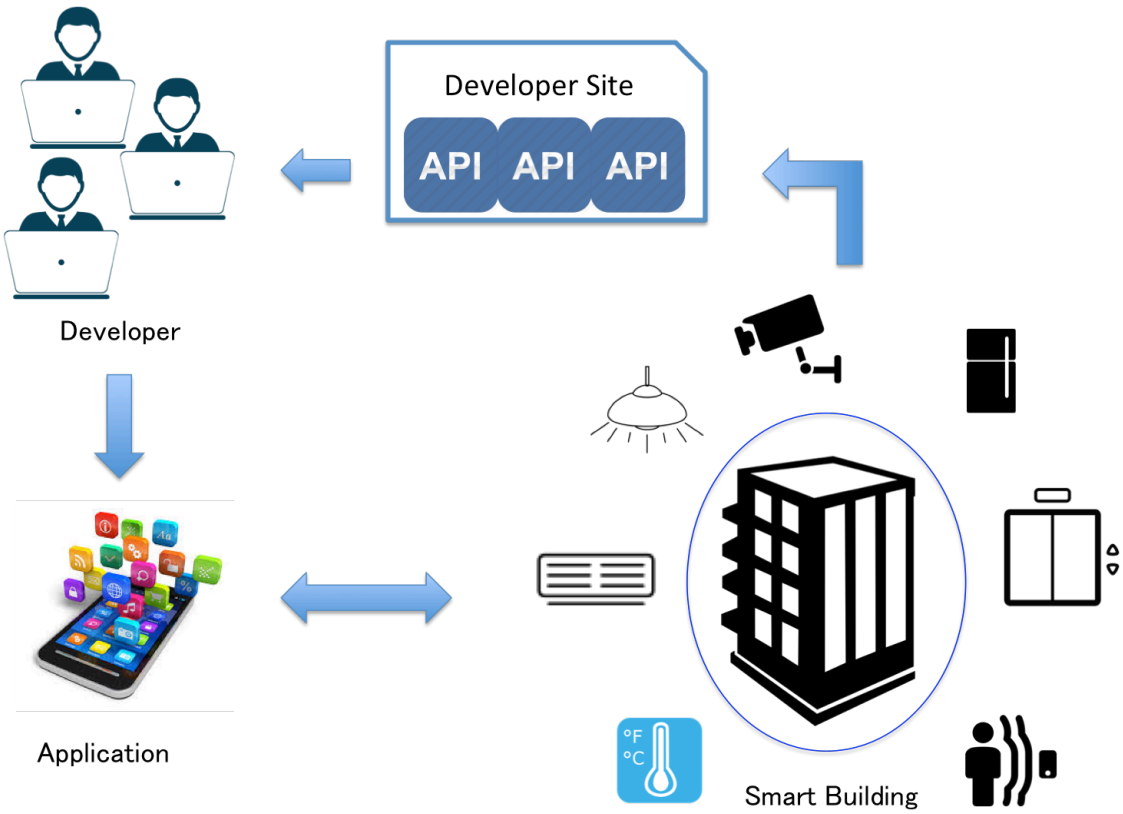


Figure 3.1: General Flow of Application Development

in the room through their terminals. To build this application, developers at least have to provide two information, the list of devices in selected room, and the list of device properties, which can be monitored or controlled by the occupant. The first information can be provided by registering where a device will be placed initially in the system. Therefore, API for obtaining the list of devices can be designed easily. However, providing the latter information is challenging because of the heterogeneity of devices in buildings. Each device has its properties, which are distinct each other. For instance, an air conditioner may have properties such as power (on or off), current temperature, set temperature, fan direction, fan speed, and etc. Different to an air conditioner, light may have properties such as power (on or off), brightness, color, and etc. Other devices such as elevator, surveillance

camera, sensors, actuators may have their own properties, which different to air conditioner and lights. Moreover, the same type of devices, which are produced by a different manufacturer, may also have different properties.

Currently, each manufacturer defines specification and design API for their devices product separately. To combine IoT devices freely and build new applications, developers have to master all of specification and configuration of device APIs they want to use. It is significant to investigate how to simplify heterogeneous device and design a de-facto standard API for IoT devices.

Due to its simplicity, RESTful architecture [17] is widely adopted in web services. In the traditional RESTful style, the property of resource is usually used as the endpoint of URI. When adopting RESTful API for IoT devices, the property of device become the endpoint of URI, for instance, the temperature of an air conditioner in a room, may describe as URI below:

`http://.../room/:id/air_conditioners/:id/temp`

The brightness of a light in a room, may described as URI below:

`http://.../room/:id/lights/:id/brightness`

To provide the list of device properties, which can be monitored and controlled by the occupant, developers have to refer the developer site to confirm the functionality of devices. For example, occupants are able to monitor and control power, temperature, and fan direction of an air conditioner. To control a power, developers have to confirm which API to be used, and what a parameter to be set when sending an invocation to the device as well as to control a temperature and a fan speed. The lack of functionality description in traditional RESTful style can lead to increase application development cost. Since a number of IoT devices are expected to grow rapidly, it is necessary to investigate a new approach to extend the traditional RESTful style and provide an appropriate design of device APIs.

3.2 API Design

To deal with the problems described in the previous section, we propose a de-facto standard of API design for open services in smart buildings. In our design, we divide the API into two categories: device API, and context API. In this section, first, we explore the structure of smart buildings to identify the entity that exists in smart buildings. Second, we explain how to standardize device API through device abstraction. Third, we discuss how we compose context API based on the relationship of entities in smart buildings which described in the first subsection.

3.2.1 Smart Building Structure

In order to design a de-facto standard of API, we need to know the structure of a smart building. First, we observe the component of smart building and the relationship between them. We also discuss how to integrate all of the entities to the smart building systems. The smart building consists of three main entities: space, occupant, and device. Space is an area inside the building. If the building is the college, it may be a classroom, a laboratory, a corridor, a lounge, a toilet, and etc. To integrate space to smart building systems, we can utilize location information technologies such as RFID, Infrared, Wi-Fi, and Bluetooth Low Energy (BLE). An occupant is an individual who occupies in the building. It may be a student, a professor, a staff, a guest, and etc. The occupant is integrated to smart building systems through their smartphone or PC. The device is including appliances, sensors, actuators, which place in entire of the building. Since they are network-enabled devices, they have been connected to smart building systems.

Since all of the services delivered in the smart building will be related to these three entities, we investigate the relationship between them. From their relationship, we found what kind of information should be defined to provide services in the smart building.

- *space-occupant.* In the relationship space-occupant, first, we observe from the perspective of *space*. At least we have to define two relations: who is allowed

to use that space, and who are currently exist in that space. Otherwise, in the perspective of *occupant*, which space is allowed to be used, and the current space of the occupant have to be defined.

- *space-device*. In the relationship space-device, from the perspective of *space*, we have to define what devices are in the space. From the *device* perspective, we have to define the current space of the device.

- *occupant-device*. In the relationship occupant-device, from the perspective of *occupant*, we have to define which devices are allowed to be used by the occupant. From the perspective of *device*, we have to define who are allowed to use the device.

3.2.2 Device API

To standardize device API, we have to accommodate all of IoT devices and simplify the properties of such devices. Based on the exploration of device property, we propose the device abstraction by classifying properties of devices into two groups:

- *Static Property*. Static property of device means the fixed one, including id, name, current place, functionality, and etc. In this work, we define the static property as *attribute*.

- *Dynamic Property*. Dynamic property of device means the property which can be changed, including the power, turn on or turn off, for air conditioners it may the temperature, for the elevator it may the floor, end etc. We define the dynamic property of device as *state*.

Basically, in our device API, we adopt the RESTful style, where the identification of devices relies on Uniform Resource Identifiers (URI) and four main methods of HTTP (GET, PUT, POST, and DELETE) are used to conduct the operation to devices. We extend the RESTful style by designing the static property, *attribute*,

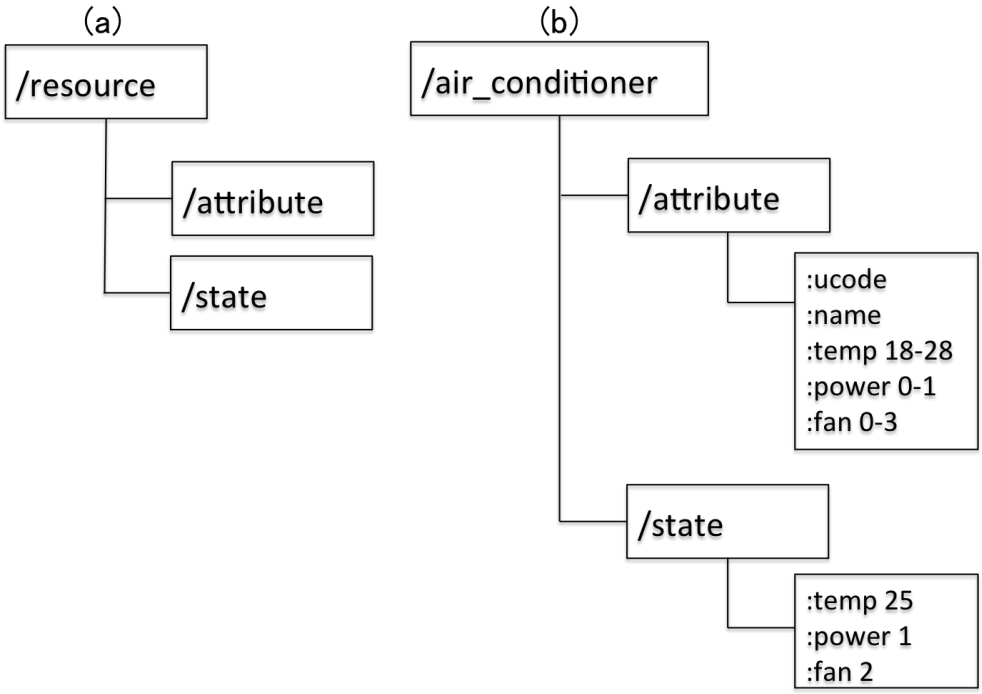


Figure 3.2: Abstraction on Device API

and the dynamic property, *state* to become the endpoint of URI. The Fig. 3.2 (a) shows the basic design of device API that proposed, and (b) shows the example of the API of air conditioners .

The Fig. 3.3 shows the comparison of endpoint between the traditional REST style and the proposed design. In the traditional RESTful style, resource's properties are designed to become the endpoint of URI. As shown in the Fig. 3.3 (a) , if the devices is defined in traditional style, we will have at least five URIs:

`http://.../room/:id/air_conditioners/:id/ucode`

`http://.../room/:id/air_conditioners/:id/name`

`http://.../room/:id/air_conditioners/:id/temp`

`http://.../room/:id/air_conditioners/:id/power`

`http://.../room/:id/air_conditioners/:id/fan`

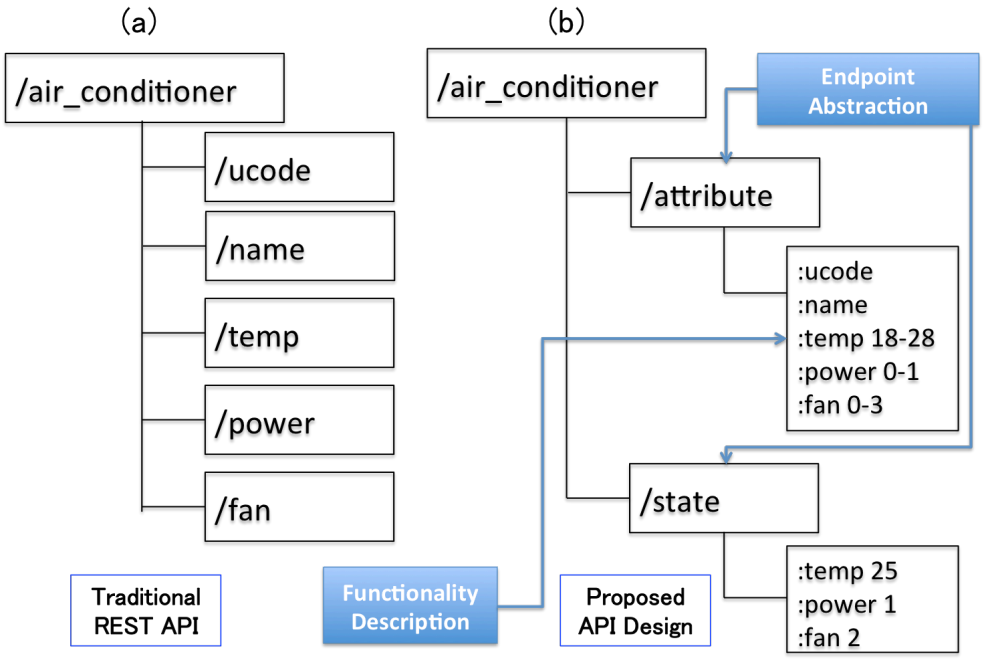


Figure 3.3: Comparison of API Endpoint

In contrast to the traditional style, we have proposed each device only have two endpoints: *attribute*, and *state* as shown in the Fig. 3.3 (b), with the simple URIs:

`http://.../room/:id/air_conditioners/:id/attribute`

`http://.../room/:id/air_conditioners/:id/state`

To overcome the lack of functionality description, we observe the useful information for describing devices in the development process. Based on our observation, the information such as the property, and parameters to set the property are useful in the development process. Since we can classify this information as the static property, we include them in *attribute*. For example, if there is an air conditioner with three dynamic properties: power, temperature, and fan speed, we can design device APIs as shown in the Fig. 3.2 (b). The list of dynamic properties and parameters, the temperature can be set from 18 Celsius to 28 Celsius, and the power can be set 0 as off, or 1 as on, and the fan speed can be set from 0 to 3, are included

Table 3.1: List of Device APIs

Device	Method	URI
Air Conditioner	GET	$/air\_conditioners/:id/attribute$
Air Conditioner	GET	$/air\_conditioners/:id/state$
Air Conditioner	PUT	$/air\_conditioners/:id/state$
Light	GET	$/lights/:id/attribute$
Light	GET	$/lights/:id/state$
Light	PUT	$/lights/:id/state$
Elevator	GET	$/elevators/attribute$
Elevator	GET	$/elevators/state$
Elevator	PUT	$/elevators/state$
Sensor	GET	$/sensors/:id/attribute$
Sensor	GET	$/sensors/:id/state$
Smart meter	GET	$/smartmeters/:id/attribute$
Smart meter	GET	$/smartmeters/:id/state$

in *attribute*.

3.2.3 Context API

Only device API is not enough to allow development of a new application and provide services, the context information such as, the current places of occupants, the list of devices exist in the room, and the list of devices allowed to be used by occupants is needed. To deal with this requirement, we proposed context API. The context API is composed based on the observation of smart building entities and the relationship between them that we have described in the previous subsection.

3.3 Evaluation

In order to investigate the feasibility of the proposed design, we implement Smart Building API in the real building named "Daiwa Ubiquitous Computing Research

Building" [78]. To make the APIs available to developers, we also publish the documentation of APIs in developer site. To ensure that occupants could only monitor and control devices they were permitted to, we have deployed access control manager in our system.

3.3.1 Implementation of Smart Building API

In our building, we have more than 300 network-enabled devices including air conditioners (AC), lights, smart meters, elevators, temperature sensors, humidity sensors, and human detection sensors. We used Ubiquitous ID (uID) architecture [25], and assigned a 128bit unique id to devices in the building. By adopting uID architecture, we can deal with the management of semantic knowledge of heterogeneous devices in smart buildings through the ucR model. For instance, the relationship of device and property, device and room, device and occupant, can be represented properly. Table 3.1 shows device API we have implemented based on the proposed API design. The implemented API can be classified into three types, GET attribute, GET state, and PUT state. GET attribute is used to obtain static properties of the device. GET state is used to obtain dynamic properties of the device. PUT state is used to change dynamic properties of the device. When sending an invocation, the users have to specify device's id in URI, and attach an access token in HTTP header. The details of access token will be explained in the chapter 4. The implementation of API for each device was carried as follows:

- **Air Conditioner**. By requesting GET attribute for air conditioners, as a response the users will obtain static property such as ucode, unique id for devices, device's name, and device's type. Moreover, the users also obtain the functionality description. Since the users are able to control power, temperature, fan speed, and fan direction of air conditioners, we design each items and range of the parameter to be added as static description. For example, to allow the users for controlling the power of air conditioners, we added the

functionality description as 0 for turn off and 1 for turn on. By requesting GET state, the users will obtain the current status of the requested air conditioner such as power, temperature setting, fan speed, and fan direction. By sending PUT state, the users are able to change the current state. In our implementation, the users are able to set power, temperature setting, fan speed, and fan direction.

- **Light**. In lights, by requesting GET attribute, the users will acquire static property of specified light including ucode, name, type, and room. In our implementation, the users are able to control light in one room trough two ways: control entire lights or per row. Therefore, we defined type to specify the requested light are group of light for one room or a single row. By invoking GET state, the users will obtain the current status of requested light. By delivering PUT state, the users could control the status of specified light.

- **Elevator**. For elevator, by sending GET attribute, the static property of elevator is obtained. By requesting GET state, the current position of elevator is acquired. By doing invocation of PUT state, the users are able to call elevator by including the position as a parameter.

- **Sensor**. In our building, there are three types of sensors: temperature sensor, humidity sensor, and motion sensor. For sensors, we only implemented two end points: GET attribute and GET state because the users could not change the current status of sensors manually. By requesting GET attribute, the users will obtain the information such as ucode, name of sensor, type of sensor and space where the sensor is placed. By sending GET state, the users will acquire the latest data from specified sensor.

- **Smart Meter**. Smart meter in our building was installed in every room. Similar to the sensor, we only implemented two end points: GET attribute and GET state because of the natural function of smart meter as a sensing device. By sending GET attribute, the users will get the information includ-

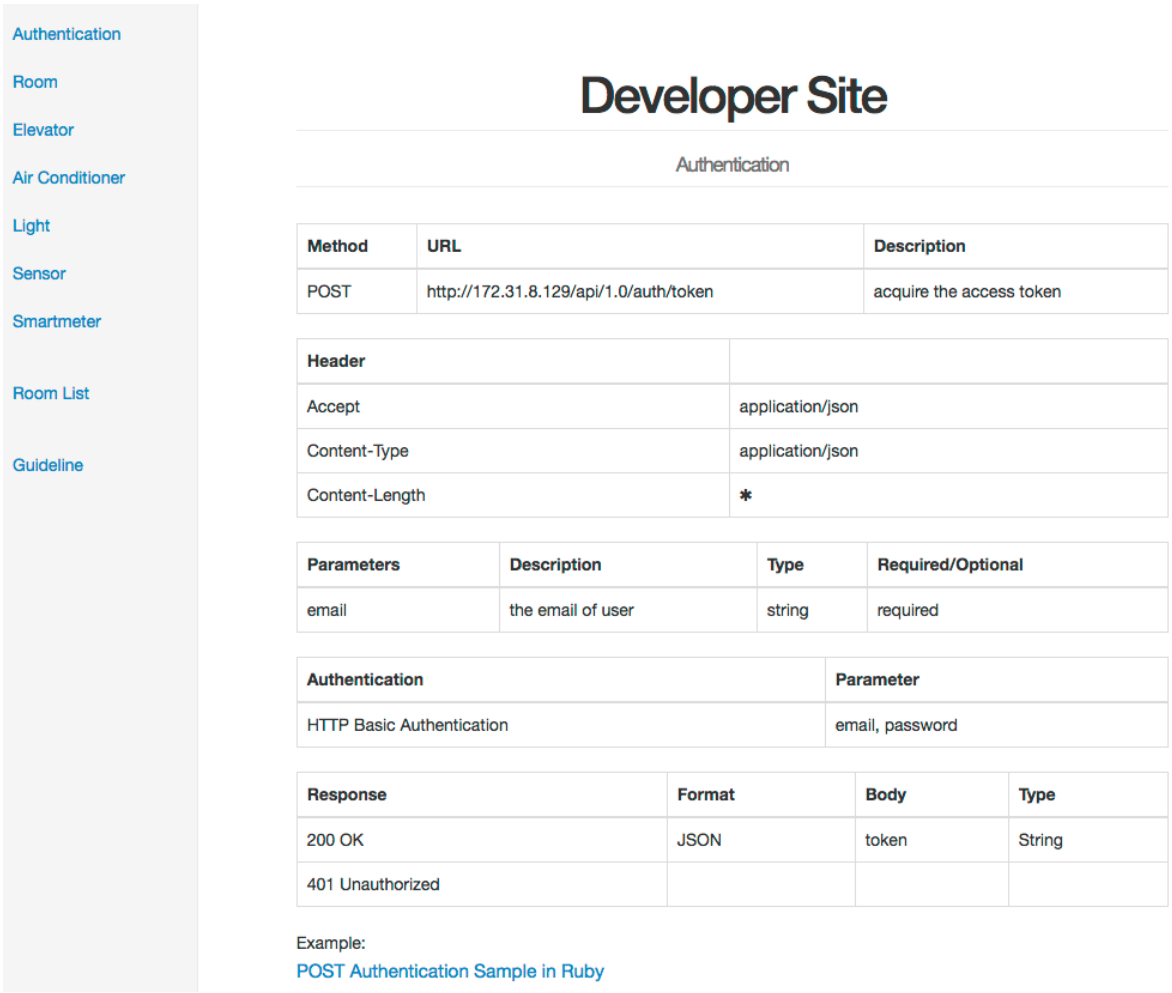


Figure 3.4: Part of Developer Site-1

ing ucode, name of smart meter, and room where smart meter is installed. By requesting GET state, the users will obtain electricity used in specified room. In order to support the flexibility, we allow the users to set parameters such as hour, day, week, month, year to specified the range of electricity used to be acquired.

The API server was implemented in 8 core Intel(R) Xeon(R) CPU E3-1230 v3 3.30GHz, with 8 GB RAM. The installed OS is Centos 6.5 and nginx server is used. We implemented Smart Building API used the Ruby on Rails framework [79].

Authentication

Room

Elevator

Air Conditioner

Light

Sensor

Smartmeter

Room List

Guideline

Method	URL	Description
PUT	http://172.31.8.129/api/1.0/air_conditioners/:id/state	Change state of the air conditioner

Header	
X-UIDC-Authorization-Token	token

Parameters	Description	Type	Required/Optional
id	id of the air conditioner	String	required
power	0: Turn Off, 1: Turn On	Integer	Required
set_temp	Temperature Setting	Float	Optional
fan_speed	0: Low, 1: Middle, 2: High	Integer	Optional
fan_direction	0: Position-0, 1: Position-1, 2: Position-2, 3: Position-3, 4: Position-4, 7: Swing	Integer	Optional

Response	Format	Body	Type
200 OK	JSON	ucode	String
		name	String
		type	String
		power	Integer
		set_temp	Integer
		room_temp	Float
		fan_speed	Integer
		fan_direction	Integer
401 Unauthorized			

Example:
PUT Air Conditioner State Sample in Ruby

Figure 3.5: Part of Developer Site-2

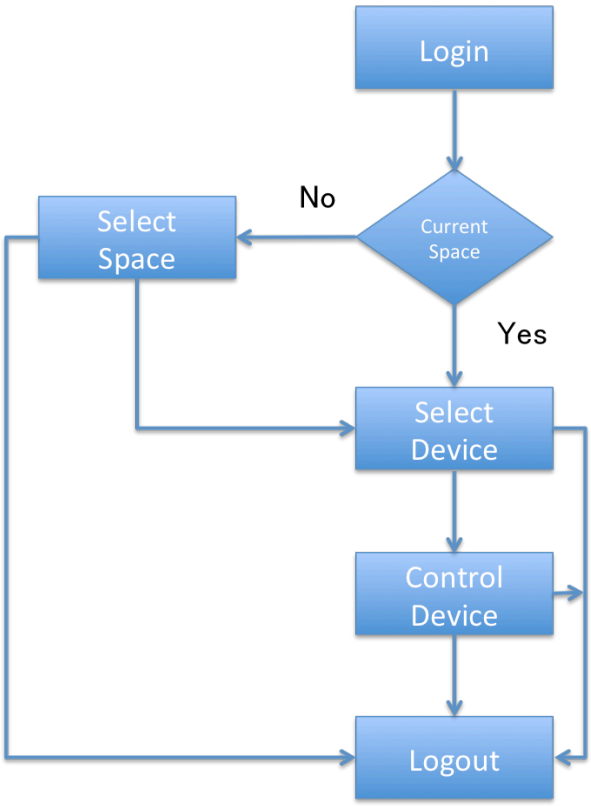


Figure 3.6: Flow Diagram of Smart Room Application

3.3.2 Developer Site

In order to publish the implemented API, we also developed Developer Site. The Fig. 3.4 and 3.5 show part of page of developer site. In the developer site, in addition to device APIs, we also publish the specification of authentication API and context APIs. Moreover, we also provided a sample code for requesting each API writing in ruby.

3.3.3 Smart Room Application

We have developed a smart room application, which can be used by occupants to monitor and control devices through their terminals such as a tablet, a smartphone, and etc. We developed a smart room application as a web application. The Fig.

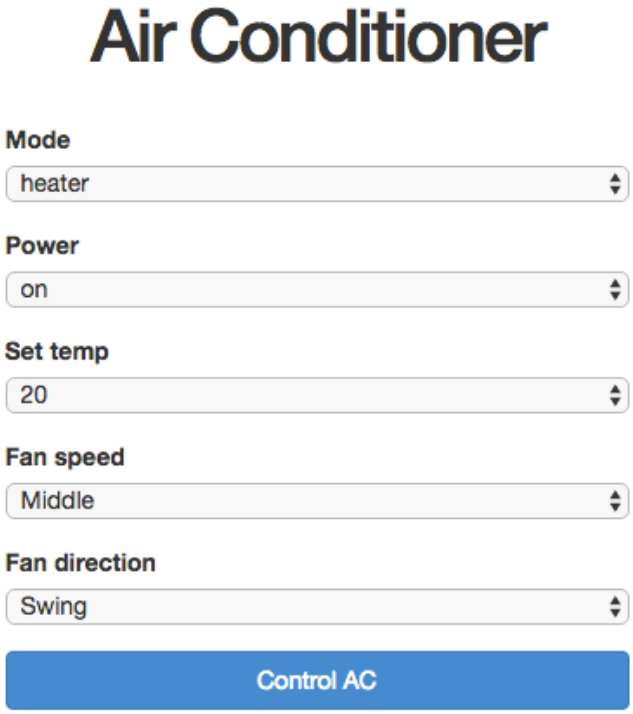


Figure 3.7: Air Conditioners Properties

3.6 shows the flow diagram of the smart room application. We compare while developing this application using the proposed API design and using the traditional RESTful style.

1. The first page of the application is a login page. To create this page, we have to look at the developer site to confirm how to process user authentication and request an access token. Either use the traditional RESTful style or the proposed design, we have to look at the developer for one time.

2. After the user successful to verify their credentials and obtain the access token, the next process is application ask the system regarding current space of the user through context API. In traditional RESTful style, we have to found the device which provides service to detect the current space of the user. In contrast, in our proposed design, we use an API to obtain user

state which the current space of user is included. In this case, the number of looking developer site will be higher when using traditional RESTful style.

3. If the current space of the user exists, the application will load a device selection page. If the system does not know the current space of the user, the application will load a space selection page. In this page, we load a list of a room which can be entered by the user. In proposed design, we can obtain this information through API of user attribute. In traditional RESTful style, this information can be included in policy. When the user selects a room, the application has to notice the system by updating the state of the room through API of room state.

4. The next page is a device selection page. The application has to display a list of devices in the current room. In the proposed design, the information of a list of devices can be obtained through API of room attribute. We have designed the attribute of room including the list of all devices in the room.

5. The last page in Smart Room Application is a device controlling page. In this page, first we display a list of functionality of a device which can be controlled by user, for example, if selected device is an air conditioner, the list may power, temperature, fan direction, fan speed, and etc. Since our proposed design including this property as a part of device attribute, we just call an API of device attribute to obtain these lists. In contrast, in traditional RESTful API, we have to check the functionality of selected device at the developer site. Since each device has different function and specification, the number of referring developer site will be higher than the proposed design.

To confirm how much application developing time can be shortened by the proposed design, we have conducted experiments for development process 5). We asked five students, which have an experience in application programming to become a developer and create a page to display a list of device properties, which can be

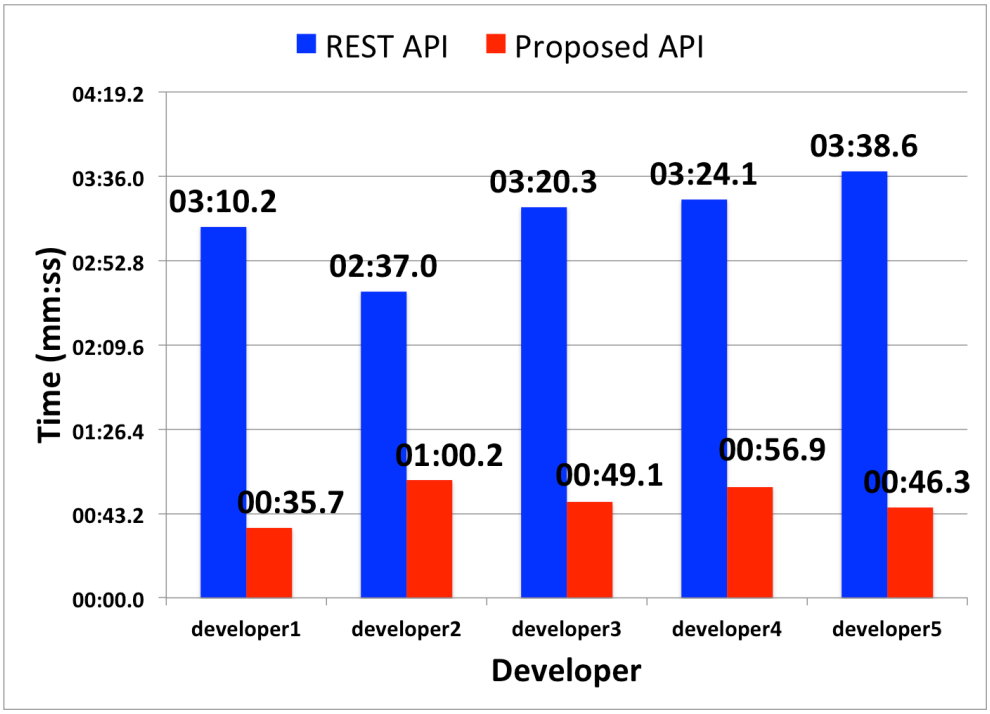


Figure 3.8: Comparison of Developing Time

controlled by occupants as shown in the Fig. 3.7. We measure the developing time when using the traditional RESTful style and using the proposed API design.

As can be seen from the Fig. 3.8, compare to the traditional RESTful style, the application development time can be shortened by referring the proposed API design for all developers. When creating device controlling page, to display a menu for controlling device, the developers have to know the properties which can be controlled by the user. By referring the traditional REST API, the developers will check manually each property to display including the range of parameters. For instance, to set the temperature to the air conditioner, the developers have to look the developer site to refer the range of temperature can be set to the air conditioner, type of parameters to be sent, and etc, which consume more time. As a result, it increases the cost of application development.

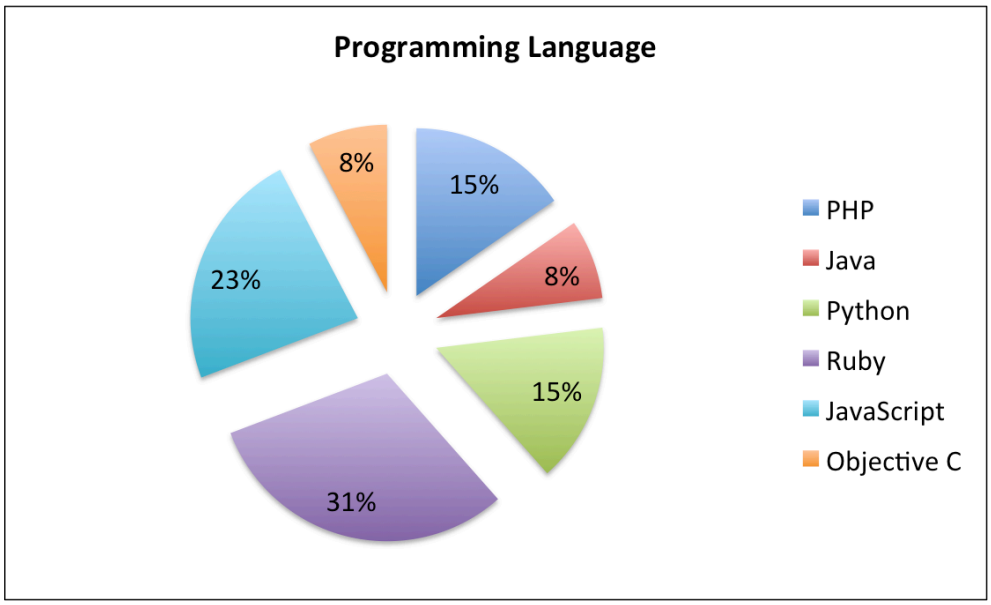


Figure 3.9: Programming Language

Conversely, by referring the proposed API, the properties which can be controlled by the user, can be generated automatically by requesting the static property endpoint. We have designed to include the useful information in development processes, such as the functionality and the parameters to set in each functionality, into the static property. Therefore, the developers can be obtained this information easily by requesting the static properties and generate the menu automatically as can bee seen in the Fig. 3.7. As the average of developing time decreased by 73.9 % by using the proposed API design, it is indicated that the proposed design is successful to reduce the development cost, which allows developers to create new applications more easily.

3.3.4 User Feedback

In order to evaluate the design of the proposed API, we have asked the users who have used the proposed API to answer the questionnaire. The questionnaire is composed of four parts. In the first part, we asked on the experience of the

participants in using web API. In the second part, we asked the questions regarding design of the proposed API. In the third part, we asked relating the developer site. In the last part, we asked about the sample code provided at the developer site. There are six users who participated in this questionnaire.

- **Web API Experience**. In this part, we asked if the users have experience in web API and what programming language that they used to. All of the participants have an experience in web API programming. The Fig. 3.9 shows the programming language that users have used to. We found that 31 % of the participants have the experience to use web API in Ruby.

- **Device API**. In this part, we asked the participants to rate four statement using a five-point Likert scale (1= Disagree strongly, 5= Agree strongly). The list of statements, mean values, and standard deviations are shown in Table 3.2 $\sim$ 3.8. The results show that the proposed APIs including the specifications are easy to use in general.

Table 3.2: Questionnaire Results on Authentication API

Statement	Average rating (SD)
Easy to understand	4.17 (0.41)
Easy to use	4.33 (0.52)
Enough parameter	4.67 (0.52)
Parameter easy to use	4.33 (0.52)

Table 3.3: Questionnaire Results on Room API

Statement	Average rating (SD)
Easy to understand	4.33 (0.82)
Easy to use	4.5 (0.55)
Enough parameter	4.67 (0.52)
Parameter easy to use	4.67 (0.52)

Table 3.4: Questionnaire Results on Elevator API

Statement	Average rating (SD)
Easy to understand	4.5 (0.71)
Easy to use	4.5 (0.71)
Enough parameter	4.5 (0.71)
Parameter easy to use	4 (0)

Table 3.5: Questionnaire Results on Air Conditioner API

Statement	Average rating (SD)
Easy to understand	4.6 (0.55)
Easy to use	4.6 (0.55)
Enough parameter	4.6 (0.55)
Parameter easy to use	4.8 (0.45)

Table 3.6: Questionnaire Results on Light API

Statement	Average rating (SD)
Easy to understand	4.4 (0.55)
Easy to use	4.4 (0.55)
Enough parameter	4.2 (0.84)
Parameter easy to use	4.2 (0.84)

Table 3.7: Questionnaire Results on Sensor API

Statement	Average rating (SD)
Easy to understand	4.5 (0.58)
Easy to use	4.5 (0.58)
Enough parameter	4.75 (0.5)
Parameter easy to use	4.5 (0.58)

Table 3.8: Questionnaire Results on Smartmeter API

Statement	Average rating (SD)
Easy to understand	4.33 (1.5)
Easy to use	4.33 (0.58)
Enough parameter	4.67 (0.58)
Parameter easy to use	4.33 (0.58)

- **Developer Site**. In this part, we asked the participants to rate two statements using a five-point Likert scale (1= Disagree strongly, 5= Agree strongly). The list of statements, mean values, and standard deviations are shown in Table 3.9. The results show that the developer site is easy to understand, and have enough explanation.

Table 3.9: Questionnaire Results on Developer Site

Statement	Average rating (SD)
Easy to understand	4 (0.89)
Enough explanation	4 (0.89)

- **Sample Code**. In this part, we asked the participants to rate three statements using a five-point Likert scale (1= Disagree strongly, 5= Agree strongly). The list of statements, mean values, and standard deviations are shown in Table 3.10. The results show that the sample codes provided are helpful and easy to use. Since the current sample codes are only in Ruby language, one participant commented it is better to give sample codes in several languages.

3.4 Summary

In this chapter, we have proposed an alternative approach to a de-facto standard API design for open services in smart buildings. To deal with the heterogeneous devices, we proposed device abstraction. Based on the investigation of device

Table 3.10: Questionnaire Results on Sample Code

Statement	Average rating (SD)
Helpful	4.83 (0.41)
Easy to understand	4.5 (0.84)
Easy to use	5 (0)

property, for each device, we defined two properties: *attribute*, static property, and *state*, dynamic property. To provide device description, we investigated the useful information in developing process and included it into the static property. To evaluate the proposed design, we have implemented Smart Building API, which is designed by the proposed design. We also developed a Smart Room Application and compare the development process between the traditional RESTful API and the proposed design. The results showed that our proposed design has been successfully reducing the development cost of new applications, and can be easily used to create new applications.

Chapter 4

Access Control Framework

In this chapter, we describe an access control framework for open services in smart buildings. First, we describe the importance of security for open services. Then, we present design of access control framework for securing open services environment. After describing the implementation of the proposed framework, we discuss how we evaluate the access control framework.

4.1 Security in Open Services

As described in the previous chapter, in order to realize the open services in smart buildings, the information associated with how to exchange data and communicate with the IoT devices have to disclose to the public. Therefore, the third developer can bring their innovation to combine various type of devices for providing new services to the occupants. As a result of publishing APIs, how to access and control the devices is known by everyone. Consequently, the security system is necessary to ensure only the authorized user can access and control the devices.

The implementation of open services without an adequate security system will cause the large security breach. It happens because the employment of IoT devices directly affects the physical environment. If an unauthorized user is able to access and control such devices, it may harm an occupant's life for instance by locking a door, turning off the lights or air conditioners, which make occupants lose their

comfortable spaces.

However, the conventional security framework from the cyber world cannot simply be adopted to the IoT ecosystem, because in IoT ecosystem like smart buildings, it may consist of heterogeneous devices, from tiny devices such as temperature sensors, humidity sensors, etc to house appliances like air conditioners, refrigerator, and etc. It is difficult to implement the complex security computations to tiny devices, which have limited computation and memory resources. Moreover, appliances and devices in smart building environment may be dynamically attached and removed at any time depend on the necessity. Hence, it is crucial to restrict the access to such devices by adopting access control framework, which handles the specific characteristics of the physical world in the smart environment.

4.2 Design of Access Control Framework

The Fig. 4.1 shows the architecture of our proposed framework. To enforcing security for API-enabled devices, security manager has two main functions: to verify that a subject is the valid user of the system and to assign an access permission of the resource. To improve the efficiency of the system, we split the security manager into two components, authentication manager, and access control manager. Moreover, by splitting the security manager our framework support the scalability of systems.

4.2.1 Authentication Manager

The main function of authentication manager is to verify user's credentials and to issue an access token for authenticated users. In our framework, before sending a request to resources, users have to retrieve access token by verifying their credentials. Table 4.1 shows the specification of authentication manager in our framework.

Users send *credentials* information as a request parameter. Users may also send *auth*, authentication method and *strength*, confidence level as optional parameters.

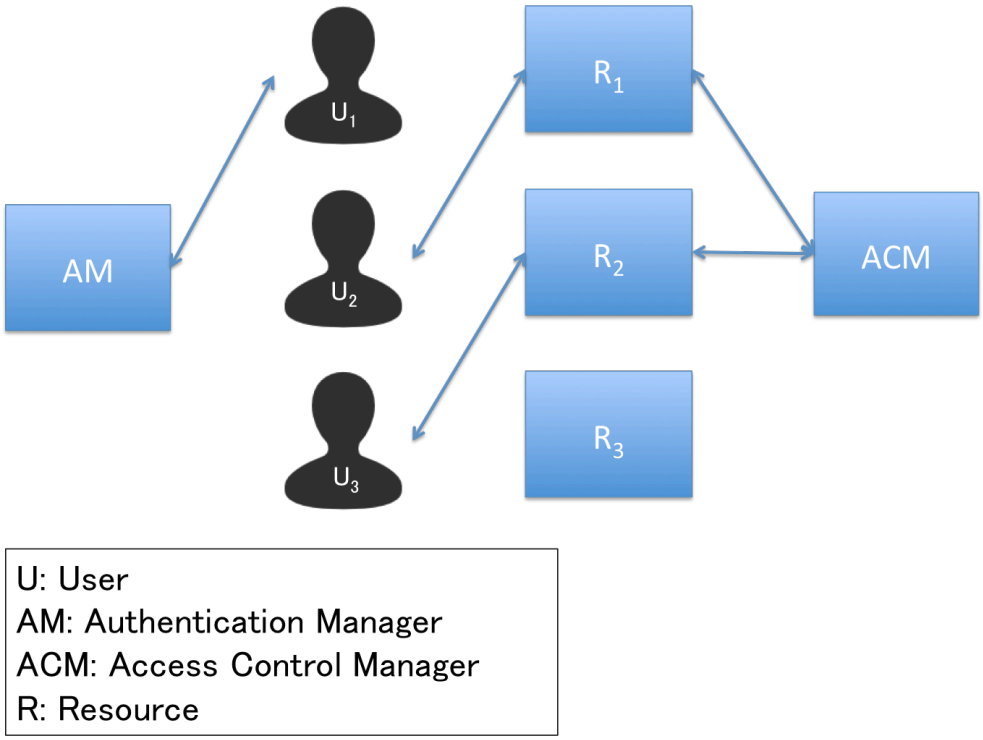


Figure 4.1: Access Control Framework

The *auth* is used for specifying which authentication method will be used to verify the users. We use *strength* to specify the lower limit of confidence level. Since the smart building environment is dynamic, we designed our framework to support various authentication methods and introduced *confidence level* [56]. The ranges of confidence level are between 0 and 1. The APIs in smart building environment have a different critical level. For example, the critical level of API for acquiring a value of temperature sensor is different with the critical level of API for controlling air conditioner. To request API, which has a high critical level, users must be verified with an authentication method, which has a high value of confidence level.

The access token will be issued for users who have been verified. The access token contains *user id, auth, strength, duration and permission* as a payload. The *duration* indicates the expiration of access token. The term of *permission* represents the authority of verified users. The permission can be adopted according to the

Table 4.1: Specification of Authentication Manager

Method	POST
URI	http://.../token
Request Parameter	credentials auth (optional) strength (optional)
Reply	200 OK 401 Unauthorized
Access Token	user_id auth strength duration permission

place of smart building system is deployed. In the college, the *permission* may be guest, student, staff, professor, and etc. This authority will be used as the parameter when to evaluate a request sending by users. In order to protect the legitimacy of the token, we attach a digital signature into the access token. The private key of authentication manager is used as a key. The legitimacy of this token can be verified by the public key of authentication manager.

4.2.2 Access Control Manager

The main responsibility of access control manager is to handle the right of users and restrict access to resources. Table 4.2 shows the specification of access control manager in our framework. In the smart building with a large number of resources and users, we need an access control, which is able to deal with these complex conditions. Since the resources in a smart building may not belong to one owner, we also need the authorization decision, which based on both attributes of resources and users. For these reasons, the policy-based access control is used in

Table 4.2: Specification of Access Control Manager

Method	POST
URI	http://.../evaluate
Request Parameter	access token resource
Reply	200 OK
Reply Body	result duration (when Permit) alternatives (when Deny)

our framework. We use XACML (eXtensible Access Control Markup Language), an XML-based language for access control that has been standardized by Technical Committee of the OASIS consortium [80]. The administrator or the owner of the resource defines security policy and save it to policy repository. With the policy-based access control, a various of access control model can be flexibly adopted. Moreover, the policy in the physical world also can be easily created.

In our framework, a verified user will send a request directly to resources. When a resource receives an invocation from a user, the existence and the expiration of access token will be checked. If the access token exists and does not expire, the resource side is forced to ask the access control manager for evaluating the accepted invocation. The access token and the resource attached as request parameters. After an evaluation request from the resource side is accepted, the access control manager will verify the legitimacy of the access token. The legitimacy of the access token verified with the public key of the authentication manager. *Authentication method, confidence level, and permission*, which included in the access token, can be used as authorization decision factors. For example, the request to a specific API requires a specific *authentication method*. Moreover, to access an API, which has a high critical level, such as controlling an air conditioner, calling an elevator, the user is required to acquire a high value of *confidence level*. Also, the *permission*,

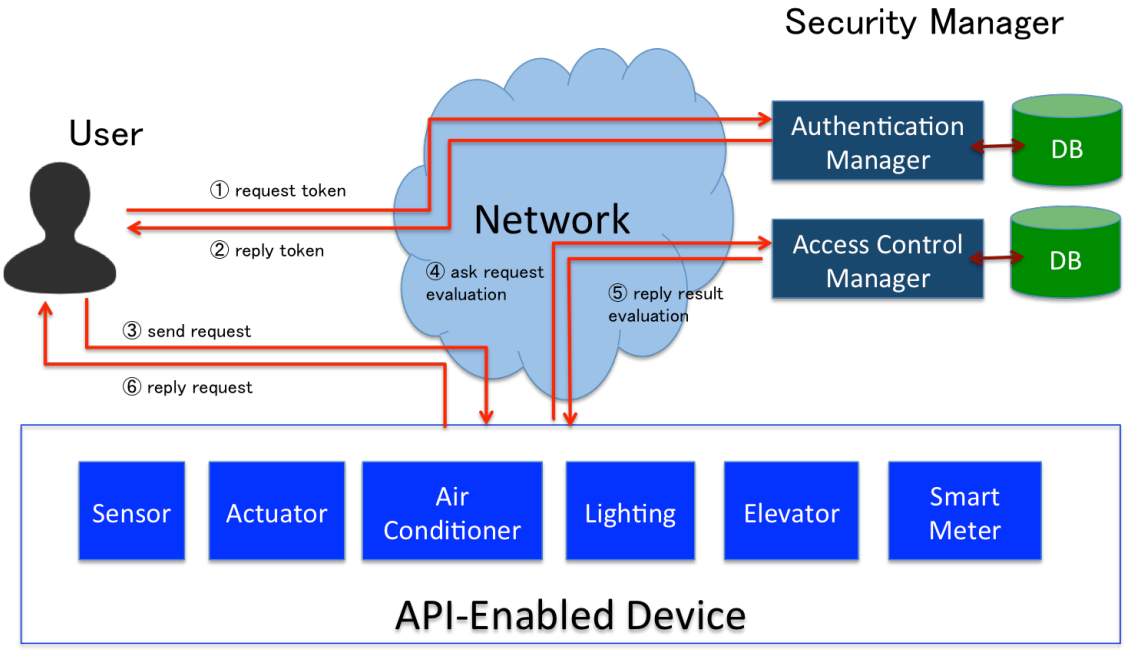


Figure 4.2: Flow of Security Enforcement

which represents the authority of the verified user, can be used as a role. Some devices can be restricted to specific roles.

The access control manager has evaluation engine, which is used to decide the authorization of requesting user. The decision will be made based on the security policies, which are composed by administrator or owner of the resource in advance, and the result will be *Permit* or *Deny*. If the result is *Permit*, reply body included the duration of permission. If the result is *Deny*, reply body included alternatives, the alternative of the authentication method. The evaluation result is sent to resource side, and resource side uses it as decision factor to serve or deny an invocation.

4.3 Implementation

To investigate the feasibility of our framework in the real smart environment, we implemented the security manager in our smart buildings system, where device APIs have been implemented. We implemented security manager as RESTful API [81] of in accordance to specification in Table 4.1 & 4.2. We implemented the security manager used the Ruby on Rails [79] framework. Firstly, we describe the implementation of user management, which can not be

The Fig. 4.2 shows the flow of security enforcement in our implementation. The user information is registered in the database. As a prototype, we use HTTP Basic Authentication for verifying the user identity and use JSON Web Token [82] as an access token. The access token is encoded by Base64 [83]. Since we implemented the security manager as REST API, it is easy to adopt other authentication methods, such as card key, biometrics, and etc.

1. The user sends HTTP POST request to authentication manager. Since we use HTTP Basic Authentication, the user has to provide a username and password. Then, the authentication manager verifies accepted username and password. If the verifying process is successful, the authentication manager issues an access token. In order to protect the legitimacy of the access token, we use RSA signature with the key size 2048 bits.

2. The authentication manager will return an access token, which contains *user id, authentication method, confidence level, token expiration and permission* as a payload.

3. The verified user sends a request to resource side. In our smart building systems, HTTP-based device API have been implemented. Through the device API, users are able to send a request like GET property, GET state, PUT state to devices such as sensors, actuators, air conditioners, lightings, elevators, and smart meters. The access token is attached to HTTP request in a new field of HTTP Header.

```
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" PolicyId="U00001C0000000000000020000000D4553-policy"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-unless-permit" Version="1.0">
 <Target>
  <AnyOf>
   <AllOf>
    <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-regexp-match">
     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">U00001C0000000000000020000000D4553</AttributeValue>
     <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
    </Match>
   </AllOf>
  </AnyOf>
 </Target>
 <Rule Effect="Permit" RuleId="U00001C0000000000000020000000D4553-professor-GET-property">
  <Condition>
   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-is-in">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">U00001C0000000000000020000000D4553</AttributeValue>
      <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource'
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
     </Apply>
     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-is-in">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">GET-property</AttributeValue>
      <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id" Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
     </Apply>
    </Apply>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-is-in">
     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">professor-Sakamura-Koshizuka-Lab</AttributeValue>
     <AttributeDesignator AttributeId="http://wso2.org/claims/role" Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
    </Apply>
   </Apply>
  </Condition>
 </Rule>
```

Figure 4.3: XACML Policy

4. The resource server checks whether the access token exists or not. If the access token exists, the expiration of the token is checked. If the token does not expire, the resource server prepares a new request to the access control manager, which contains the resource name, and the token as request parameters.

5. The resource server sends HTTP POST request to the access control manager for evaluating an accepted invocation. The access control manager verifies the legitimacy of the token with the public key of the authentication manager. If the verifying process is successful, the payload of the token is used as factors decision in evaluation engine. We use XACML policy language to define the security policy. We use WSO2 balana [84], the open source XACML

implementation, as an evaluation engine. We have composed the security policy in XACML format for every type of devices in our smart building system. The policy restricts access to the device based on the role. For this implementation, the access control manager uses the *permission* which is included in access token as a decision factor. The result of the evaluation is returned to the resource server. The Fig. 4.3 shows part of the XACML policy that we used in our implementation.

6. If the evaluation result is *Permit*, the resource server executes and reply the request. If the evaluation result is *Deny*, the resource server returns 401 unauthorized error. To improve the performance of access control manager, the cache is implemented in the resource server.

Although the concept of the token in our framework is similar with OAuth[85], which has been widely adopted in web architecture, the implementation of proposed framework is different to OAuth as showed in Fig 4.4. In OAuth, the token is issued after the owner authorizes the client to access the resource. Since the smart building consists of heterogeneous devices, which the heterogeneity of critical level of API, it is inefficient to decide which APIs are authorized to the client before issuing the token like OAuth flow. By contrast, the token in our framework is issued after the user verified the credentials and the authorization process is enforced after the user sends a request to the resource. Moreover, the token in proposed framework is designed to include user's information which is used not only for authorization process but also can be utilized, for example, to create access logs, caches, on the resource side.

4.4 Evaluation

In order to show that the proposed framework is feasible, first, we describe that the proposed framework is able to deal with the threats in smart buildings. Second, we explain that the proposed framework is usable and easy to integrate with

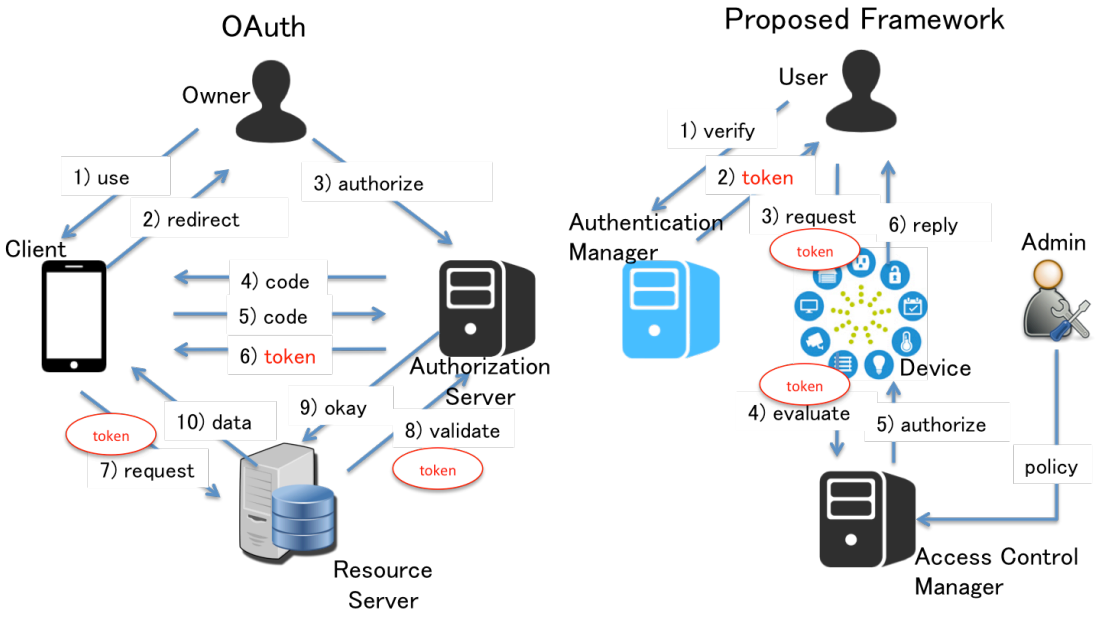


Figure 4.4: Comparison to OAuth

existing device APIs. Third, we describe that the proposed framework is scalable compared to the conventional framework. Finally, we describe the experimental result to evaluate the performance of the implementation.

4.4.1 Security

Here we describe that our framework is able to deal with the threats in smart buildings.

Unauthorized Access

As a countermeasure of unauthorized access, we adopted the access token. Everyone who tends to access the resources is forced to verify oneself through authentication manager. An intruder may also guess a password and request an access token to authentication manager. To prevent the unauthorized access, we introduce the confidence level. Despite the password of users is cracked, we can set the

confidence level of authentication method based on password with the low number and prevent unauthorized access from critical access.

Eavesdropping

In order to retrieve an access token, intruders may examine the packet between the user side and resource side. To prevent this attack, the duration is included in the access token. Although the intruder steals the access token, the user is able to re-authenticate itself and the stolen access token become expired.

Token Tampering

As a countermeasure of token tampering, we adopted the digital signature to an access token. The issued access token is signed by the private key of authentication manager. The legitimacy of the access token can be proved by the public key of the authentication manager. If intruder modified the access token, the request, which is sent by the intruder, will be denied.

4.4.2 Usability

Here we explain that our framework is usable in the real smart building. We have designed the framework to be adopted easily in the real smart building. In order to reduce the modification in the resource side when integrating our proposed framework to existing device API, we have designed the access token to be attached in the HTTP header. Therefore, the addition of an access token does not affect the current parameter of API specification. We also designed the framework as RESTful API, as a result, the developer is able to adopt the framework easily in the HTTP-based application.

4.4.3 Scalability

First, we explore the scalability of security framework when the number of devices increased. In in-situ security framework, the security manager is needed for each

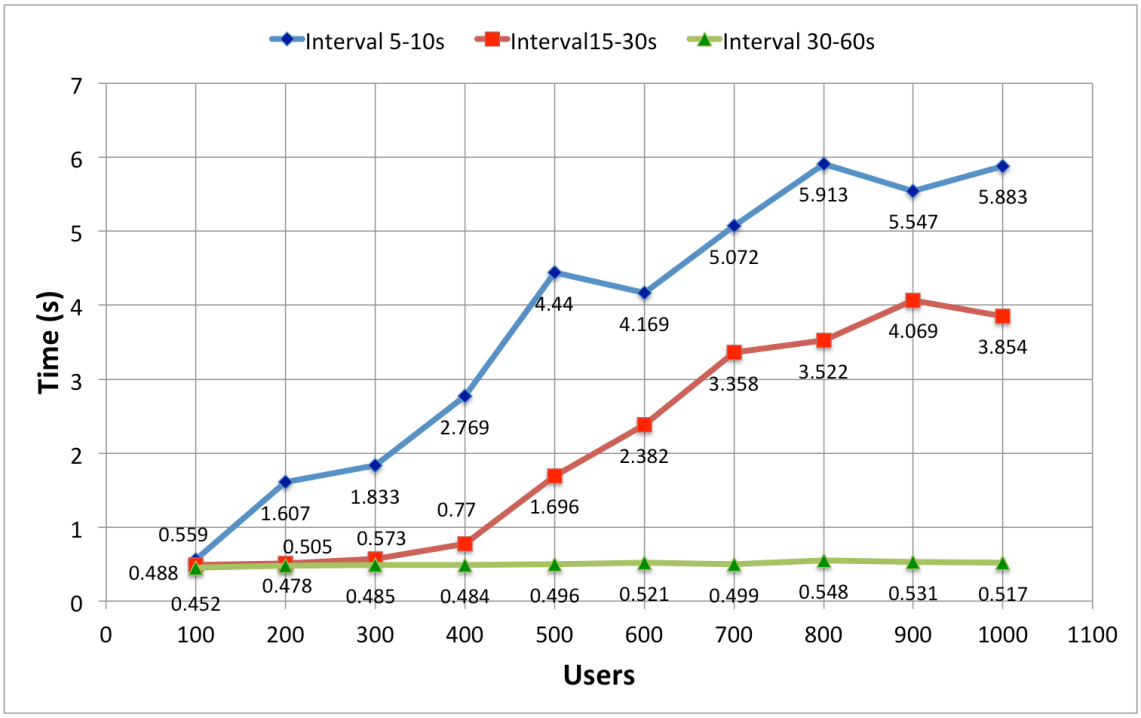


Figure 4.5: Scalability

independent new device. In proxy security framework, it is needed to modify the security manager. The information of who is authorized to access the device has to be added. In the proposed framework, it is only needed to modify the security policy. Compared to the conventional security managers, the modifying security policy is less costly.

Second, we investigate the scalability of security framework when the number of users increased. In in-situ security framework, it is needed to modify security manager on each device, which is accessible for the new user. In proxy security framework, the modification of adding a new user to the security manager is needed. In the proposed framework, it is only needed to add a new user to authentication manager.

Third, we compare the scalability of security framework when the number of policies increased. In in-situ security framework, in order to adopt a new policy,

the entire security manager in devices have to be modified. In proxy security framework, the security manager, which is the center of the system has to be modified. In the proposed framework, it is only needed to add a new policy to policy repository.

Through these comparisons, the proposed framework shows the less costly and the best scalability. Furthermore, the proposed framework also considered the heterogeneity of devices, which is difficult to implement in in-situ security framework. Moreover, the proposed framework introduces the decentralized concept of security manager by dividing it into the authentication manager and access control manager. This approach addressed the bottleneck in proxy security framework which security manager is placed in the center of the system.

In order to investigate the scalability of the proposed framework when the number of users increases, several experiments have been conducted. We conducted experiments with 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000 users. Each user is set to send a request for 100 times. The sending request was an HTTP GET request, which asked for a current state of a device. In the first experiment, to suppose a heavy load environment, we set an interval time between requests in random by 5 to 10 seconds. In the second experiment, the interval time of requests are set in random by 15 to 30 seconds. In the third experiment, to suppose a normal environment, we set an interval time between requests in random by 30 to 60 seconds. We record the response time of all requests and calculate the average of response time for each number of users. The Fig. 4.5 shows the average of response time for each number of users. In the heavy load condition, the response time increases linearly. However, the response time shows the stable value in the normal condition.

4.4.4 Performance

In order to evaluate the performance of the implementation of the proposed framework, a various of experiments have been conducted. The experiments have

Table 4.3: List of HTTP Request for API-enabled Device

Device	HTTP request
Air Conditioner	GET $/air\_conditioners/:id/state$
Light	GET $/lights/:id/state$
Sensor	GET $/sensors/:id/state$
Smart meter	GET $/smartmeters/:id/state$
Elevator	GET $/elevators/state$

been conducted in a real smart building named "Daiwa Ubiquitous Computing Research Building". In this experiment, we used 84 sensors, 114 lights, 81 air conditioners, 23 smart meters, and 1 elevator. The resource server and Security Manager was implemented in 8 core Intel(R) Xeon(R) CPU E3-1230 v3 3.30GHz, with 8 GB RAM. The installed OS is Centos 6.5 and nginx server is used.

We conducted the experiment in the client-server model. As a client, we developed an application that sends a request through device API in a range of time. We conducted experiments with 10, 20, 30, 40, and 50 users. In one experiment, each user sends a request for 100 times, and we set an interval time between requests in random by 5 to 10 seconds. The sending request was an HTTP GET request, which asked for a current state of a device. Table 4.3 shows the list of HTTP request for API-enabled devices. A device was chosen randomly for each invocation.

The Fig. 4.6 shows the result of the experiments for 10, 20, 30, 40 and 50 users. The Fig. 4.7 shows the cumulative distribution of the experiments for each 10, 20, 30, 40, and 50 users. Table 4.4 shows the average time and standard deviation for each measurement in client side. From the measurement in client side, we found that the average of response time increased as the number of users. In the experiments for 10, 20, 30 users the standard deviation almost same, less than 30 ms. On the other hand, in the experiments for 40, 50 users the standard deviation increased. In the 50 users, there were 20 requests, which have the response time

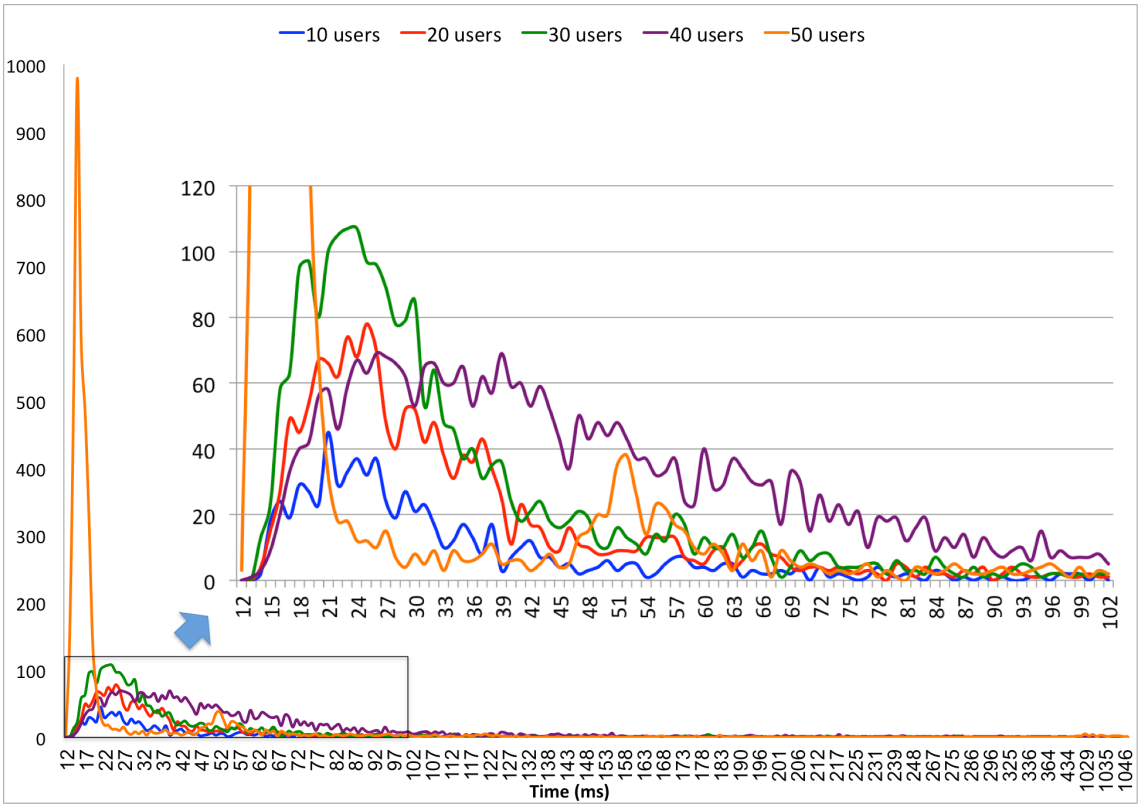


Figure 4.6: Response Time

more than 1 seconds. Since it happens at the almost same time, it may be caused by the bottleneck in resource server. It seems that the limited of users, which can be served by one resource server, one Authentication Manager, and one Access Control Manager, was 50 users. For better performance, we need to decentralize the server.

In order to investigate the scalability of the proposed framework when the number of users increases, several experiments have been conducted. We conducted experiments with 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000 users. Each user is set to send a request for 100 times. The sending request was an HTTP GET request, which asked for a current state of a device. In the first experiment, to suppose a heavy load environment, we set an interval time between requests in

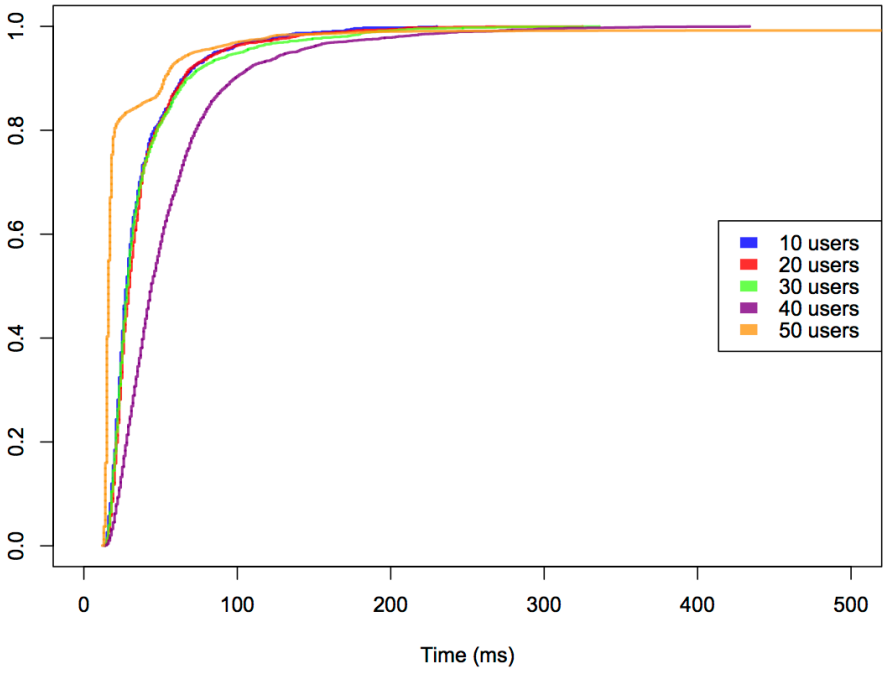


Figure 4.7: Cumulative Distribution of Client Side Response Time

random by 5 to 10 seconds. In the second experiment, the interval request time is set in random by 15 to 30 seconds. In the third experiment, to suppose a normal environment, we set an interval time between requests in random by 30 to 60 seconds. We record the response time of all requests and calculate the average of response time for each number of users. The Fig. 4.5 shows the average of response time for each number of users. In the heavy load condition, the response time increases linearly. However, the response time shows the stable value in the normal condition. It showed that the proposed framework is scalable enough in normal condition when the maximal number of users are 1000 person, and the interval request time between 30 to 60 seconds.

Our experimental results show that the average time of response time in our framework less than 0.1 seconds, which means our framework satisfy the first

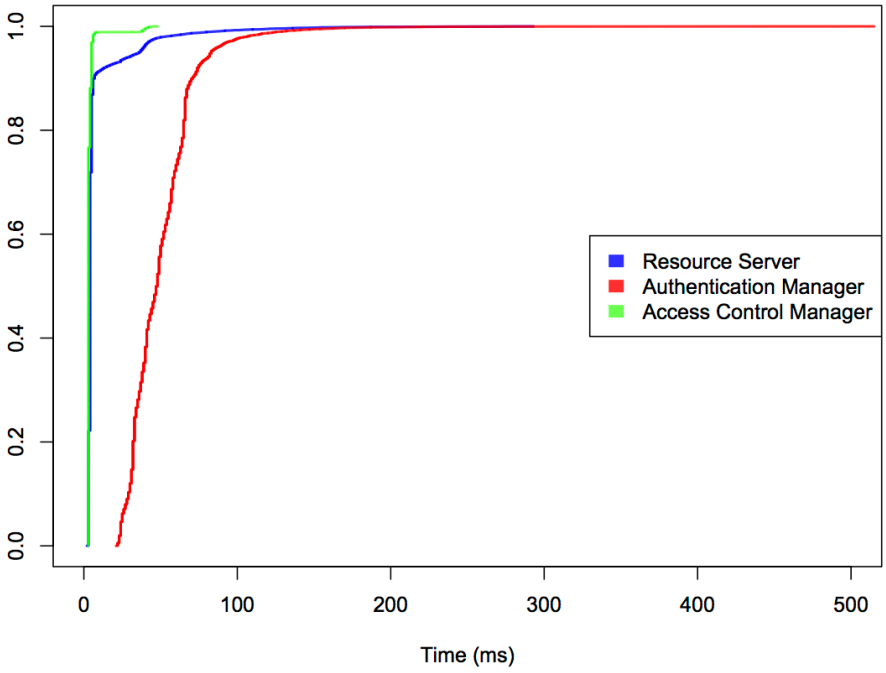


Figure 4.8: Cumulative Distribution of Server Side Execution Time

Nielsen[86] limitation. Indeed, the experimental in 50 users showed response time which over than 1 second. Overall, the experimental results indicate that our framework can be used practically in real smart buildings.

4.5 Summary

In this chapter, we have proposed an access control framework for API-enabled devices in smart buildings where devices may be attached or removed dynamically. To accommodate this kind of dynamism, we separated security manager as a third party in the system. Since smart environments like smart buildings consist of heterogeneous devices including those with limited resources, the complex security computations are delegated to the security manager. We implemented the

Table 4.4: Results of Measurement in Client Side

Number of Users	Average Time	Standard Deviation
10 users	36.78 ms	26.76 ms
20 users	38.28 ms	28.77 ms
30 users	39.26 ms	23.38 ms
40 users	56.55 ms	44.06 ms
50 users	32.47 ms	93.04 ms

Table 4.5: Results of Measurement in Server Side

	Average Time	Standard Deviation
Resource Server	7.75 ms	16.45 ms
Authentication	50.13 ms	22.01 ms
Access Control	3.76 ms	3.91 ms

proposed framework in the real building environment, which consist of more than 300 devices. Through the analysis, we showed that the proposed framework is capable of enforcing security in smart environments. To evaluate the performance of the implementation, we also conducted several scenarios of experiments. From the measurement both client side and server side, the result showed that the average of response time less than the first Nielsen limitation, 0.1 seconds, indicating our proposed framework is feasible to be practically used in real smart buildings.

Chapter 5

Services in Smart Buildings

In this chapter, we present one of the concrete examples of delivering service in smart buildings by utilizing IoT devices. First, we describe the purpose of smart buildings based on the investigation of the previous research. Then, we discuss the problem of realizing comfort environment in smart buildings. Next, we propose an algorithm for predicting collective user preference in smart buildings. We also describe how we implement the proposed algorithm and evaluate it in the real smart buildings.

5.1 Purpose of Smart Buildings

The concept of smart buildings has emerged as the utilization of information and communication technology (ICT) to overcome the various problems in the building environment. For many years, the research community has addressed the issues related to building environment including energy consumption, user comfort, user safety, and etc. The rapid advancement of ICT provide a great potential to manage the building environment for example by installing wireless sensor network throughout the building, the real-time monitoring can be realized. The feedback from the environment can be used as optimization tools to make the building more efficient.

The rapid growth of IoT devices can be utilized to provide a wide range of services

to the occupants. Before presenting a concrete example of delivering service in smart buildings, first, we observe the purpose of smart buildings. Based on our investigation related to services in buildings presented in chapter 2, we can define two purposes of smart buildings:

- **To Reduce Energy Consumption**. In order to provide the reduction of energy consumption, the IoT devices can be used to improve the efficiency. For instance, by monitoring the occupant behavior, it is possible to decrease the waste energy by adjusting the Heating, Ventilation and Air Conditioner (HVAC) system to start up when necessary.

- **To Provide Comfort Environment**. Regarding the purpose of providing comfort environment, the IoT devices can be adopted to recognize the comfort condition for the occupants. Since the usage of appliances such as light and air conditioner are relating to reaching the comfort condition of the occupants, it is possible to perceive what is the comfort condition by recording the log of the usage of such appliances.

As a concrete example of delivering service in smart buildings, we investigate a service for providing comfort environment, especially on user preference in smart buildings. In the next section, we investigate the problem of predicting user preference in smart buildings.

5.2 User Preference in Smart Buildings

How to provide a comfortable environment for occupants is one of the aims of smart buildings. Since people spend more than 80% of their lives in buildings, the environmental comfort strongly affects the occupant's productivity and quality of life and work. The occupants use the device in buildings to reach their own comfort conditions. For instance, if the room temperature is too high, the occupant will turn on the air conditioner to cool down the room. If the room is dark, the occupant

will turn on the light to brighten up the room. The action such as turn on the air conditioner and the light is taken in order to satisfy their own comfort conditions. By recognizing the user preference from user behavior, it is possible to provide comfort environment.

The prediction of user preference can be performed by monitoring the usage of appliances. For example, by monitoring the usage of air conditioners, the systems are able to acquire the information related to user preference such as the setting temperature, the setting of fan speed, the setting of fan direction and etc. By analyzing the usage log of such appliances, the prediction of user preference can be conducted. The systems could provide the comfortable environment by adopting the user preference to the appliance.

However, how to provide the comfortable environment in buildings is challenging because the building spaces are occupied by a group of occupants. Adopting one user preference may not satisfy the comfort conditions of all occupants. In addition, the comfort conditions of occupants may dynamically change depend on the external environment. For instance, the comfort conditions during the winter season may different from the summer season. Therefore, it is necessary to investigate how to predict collective user preference, which dynamically changes.

5.3 Collective User Preference

In this section, we propose an algorithm for predicting collective user preference. First, we explain the basic idea on how we extract a relationship of the occupants from the interaction between the occupants and the system and used it for predicting collective user preference. Second, we describe how to resolve the conflict of the relationship between the occupants.

5.3.1 Basic Idea

We started by investigating how the occupants do an action in order to satisfy their own comfort conditions. The Fig. 5.1, 5.2, 5.3, and 5.4 show the flow of

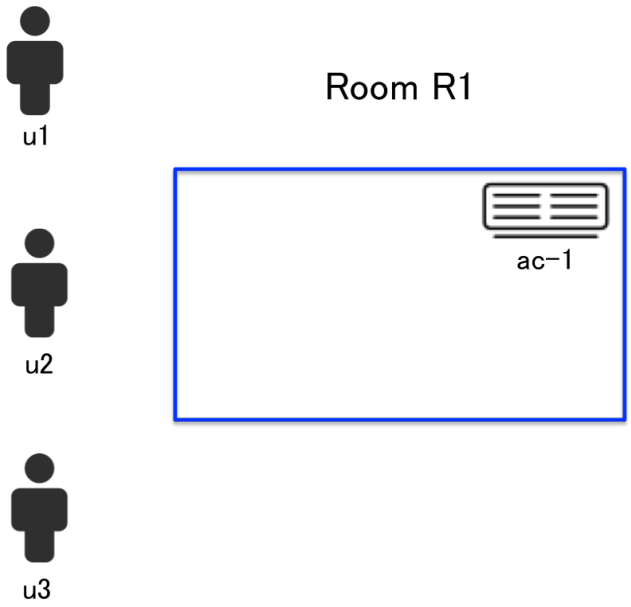


Figure 5.1: Occupants Action in The Room-1

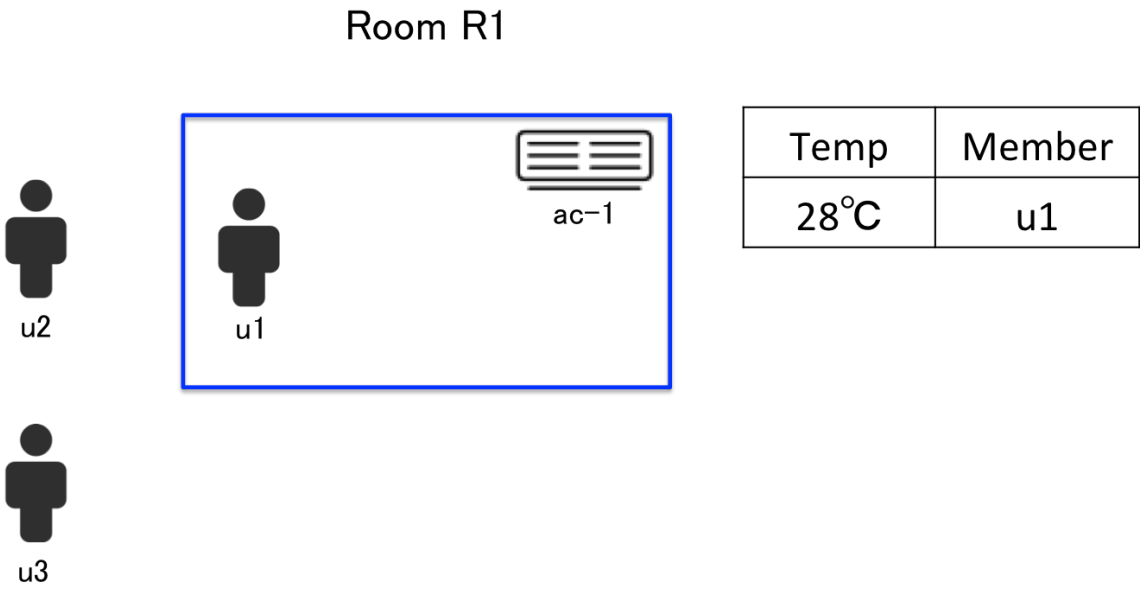
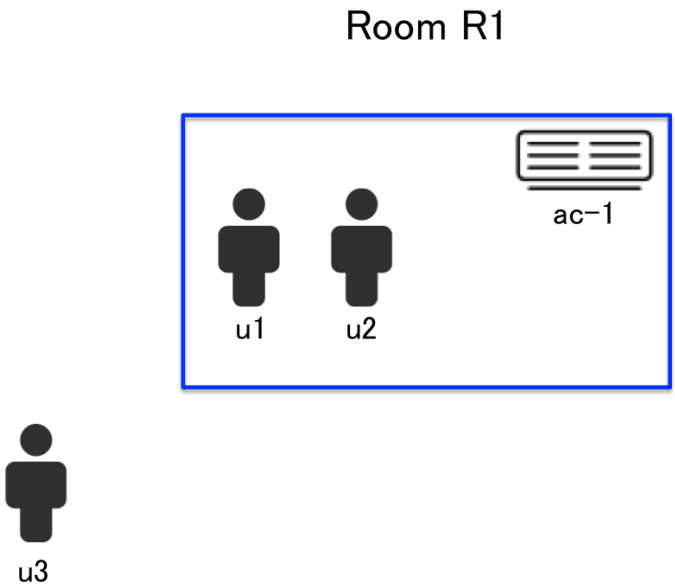
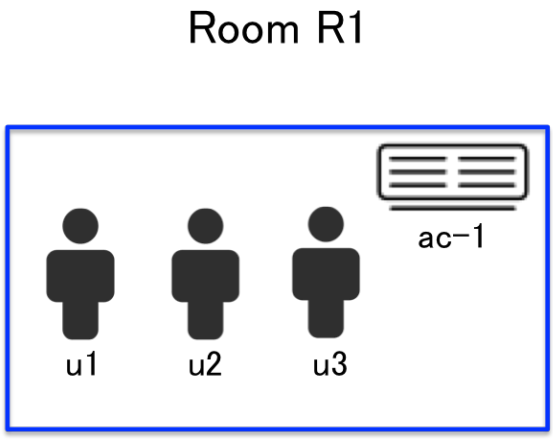


Figure 5.2: Occupants Action in The Room-2



Temp	Member
28℃	u1
28℃	u1, u2

Figure 5.3: Occupants Action in The Room-3



Temp	Member
28℃	u1
28℃	u1, u2
22℃	u1, u2, u3

Figure 5.4: Occupants Action in The Room-4

how the group of occupants do an action in the sharing spaces. In the room R1, there are three occupants u1, u2, and u3, who are sharing spaces. The room R1 has one air conditioner ac-1, which is used to control comfort in the room. As can be seen in the Fig. 5.2, an occupant u1 enters the room R1, because the current conditions do not satisfy the occupant, he would turn on the air conditioner ac-1, and set the temperature to 28°C through the system. As a result, the access log will record the member of the group is u1 and the comfort temperature is 28°C. Afterward, an occupant u2 enters the room R1, because he feels comfortable with the current conditions, he does nothing. Because the member of the group of occupants changes, the access log will record the members of the group are u1 and u2, and the comfort temperature is 28°C. Then, an occupant u3 enters the room R1, because the current conditions do not satisfy him, he does an action by changing the set temperature of air conditioner ac-1 with 22°C. Therefore, the access log will add a record, which the members of the group are u1, u2, and u3, and the comfort temperature is 22°C.

From the above observations, we noticed that the comfort conditions for a group of occupants can be known from the last manual intervention on changing conditions. In the smart buildings, the group of occupants who are sharing spaces in the room, and every manual intervention on changing conditions can be recorded in the access log. By utilizing this information, we design an algorithm for learning collective user preference that can be used to provide an optimal comfort level to the occupants in the smart buildings. Based on the above observations, we design a model and propose an algorithm.

Formally we define U as a set of all occupants:

$$U = \{u_1, u_2, ..., u_n\} \tag{5.1}$$

We define comfort condition of all occupants C as:

$$C = \{c_1, c_2, ..., c_n\} \tag{5.2}$$

where c_i is a comfort condition of u_i. Comfort condition may be room temperature,

illumination, or humidity. At first, all c_i is initialized as undefined, which indicates that the comfort condition of u_i is unknown.

When the set of users G_L are sharing space in the room, and an occupant u_L who is a member of G_L changed the room condition manually, the new condition c_L is recorded as new log entry L in access log,

$$L = \{G_L, u_L, c_L\} \tag{5.3}$$

From this log entry, we can make the following assumptions:

- Compared to the others occupants in G_L, u_L has greater authority. For example, if $G_L = \{u_1, u_2, u_3\}$ and $u_L = u_1$, the system learns that $u_1 > u_2$, $u_1 > u_3$. The notation $a > b$ means that a has greater authority than b.
- The preference of u_L is c_L, and the system updates $c[u_L] := c_L$

For example, if we have two logs $L_1 = \{G_x, u_1, c_x\}$ and $L_2 = \{G_y, u_2, c_y\}$, where $G_x = \{u_1, u_3, u_4\}$, $G_y = \{u_1, u_2, u_3\}$, $c_x = 22°\text{C}$, and $c_y = 26°\text{C}$. From G_x and u_1 we get:

$$u_1 > u_3, u_1 > u_4 \tag{5.4}$$

and from G_y and u_2 we get:

$$u_2 > u_1, u_2 > u_3 \tag{5.5}$$

We want to use the two logs above for optimizing the room temperature for new a user group $G_z = \{u_1, u_2, u_4\}$, who are sharing space in the room. We decide who has the greatest authority in G_z by using the previous logs. Considering that $>$ is a partial order relationship, we obtain the following equation from (4) and (5),

$$u_2 > u_1, u_2 > u_4 \tag{5.6}$$

which means that u_2 has the greatest authority in G_z. According to (6), we can choose c_2 as the comfort temperature for G_z.

From the observation above, we can express the hierarchical relationship of $U = \{u_1, u_2, u_3, u_4\}$ as directed graph in Fig. 5.5.

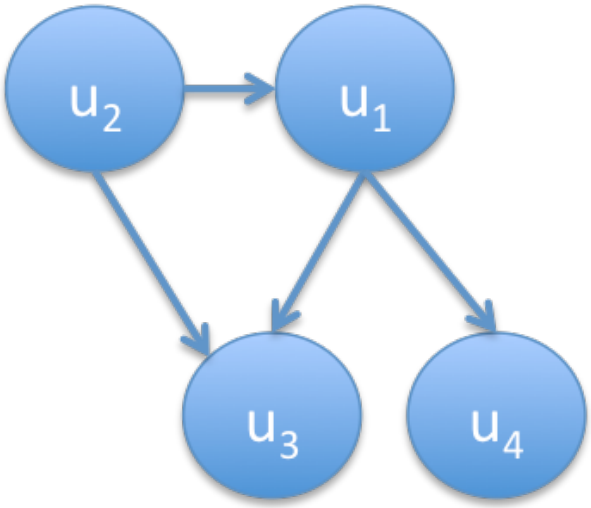


Figure 5.5: Hierarchical Relationship of Occupants

We use topological sorting [87] top-to-bottom for choosing which user's comfort preference to be adopted for all users. If the system does not know the hierarchical relationship between members of collective users, the system provides a default comfort preference and waits until an occupant who has greater authority changes the condition.

The algorithm predicts the collective user preference based on the interaction between the occupants and the system. Therefore, the algorithm can be used even in a group of occupants which do not have a hierarchical relationship. The occupants may decide the comfort conditions by voting. After they decide the comfort conditions, one of the occupants will apply it to the system. As we explain before, the system will record this condition as comfort condition for them and it can be used for predicting comfort conditions of the same group of occupants in the future.

In order to keep user preference and user relationship information up to date, we update user preference when the occupant directly accesses and change the current conditions. We also update the occupant's relationship when the room state changes, the occupant enter or leave a room, and the occupant makes a

change to the current conditions.

5.3.2 Resolving Conflicts

In the real world, the hierarchy may not exist between the occupants who share the building spaces. In this condition, each user may tend to decide the comfort conditions and do arbitrary action in order to apply their own comfort conditions. If the edge is added to the occupant's relationship graph for every action to the environment, the loop may be created. The loop will cause inconsistency of the relationship between the occupants.

We investigate the following possible solutions to prevent the loop of relationship:

- Remove the older relation. By removing the older relation, we restore the relationship to initial state and wait until the next manual intervention. It allows the system to deal with the environment, which the several occupants have the same authority.

- Keep the older relation and ignore the new one or remove the older relation and apply the new one. In this solution, the loop of relationship is prevented by keeping one relationship and ignoring the other one. However, in this solution, the system can not deal with the environment that has occupants with the same authority.

- Keep the relation which has the bigger count of access. In this solution, the loop of relationship is prevented by keeping one relationship, which has the greater number of access. However, with this solution, an occupant is able to plan to get priority from the system by increasing the number of access. As a result, the system will prioritize an occupant, who are not the user with the greatest authority.

Based on the investigation, we prevent the loop of relationship with the first solution, by only removing the old one. Therefore, the same authority of occupants can be defined and no one will get the priority.

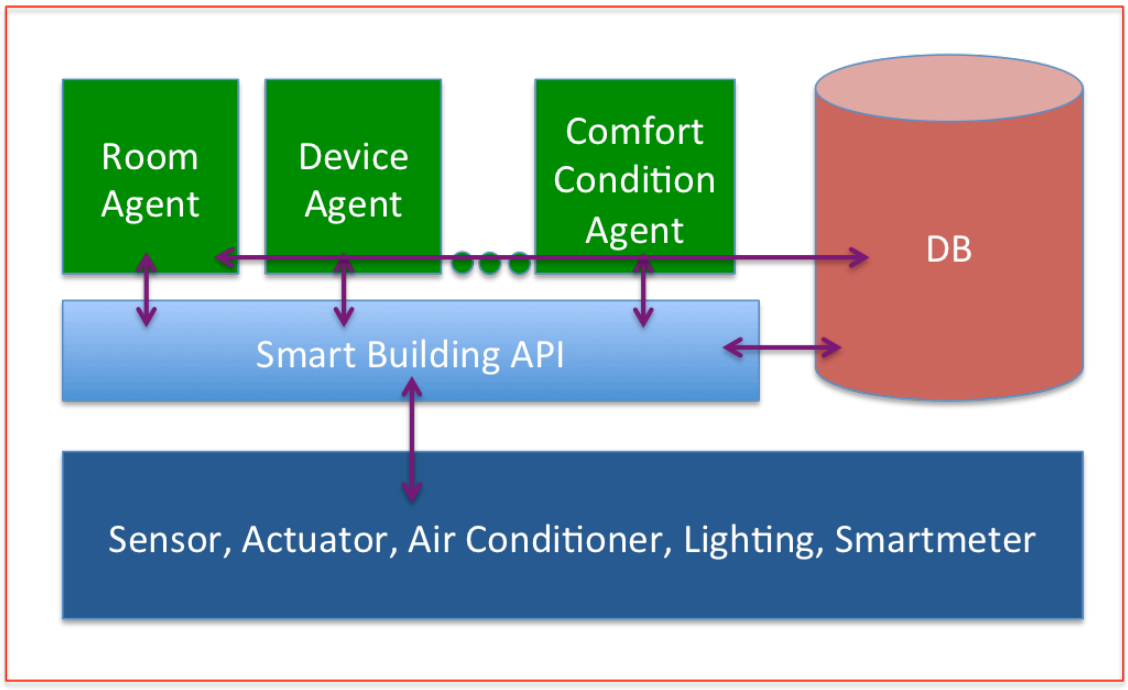


Figure 5.6: Smart Building System Architecture

In order to avoid inconsistency of the relationship of occupants, we break loops by removing the older relation that caused the loop. In the previous example, the directed graph can be denoted as below.

$$\begin{aligned} G &= \{V, A\} \\ V &= \{u_1, u_2, u_3, u_4\} \\ A &= \{(u_2, u_1), (u_2, u_3), (u_1, u_3), (u_1, u_4)\} \end{aligned} \tag{5.7}$$

If users u_1, u_2, u_3, u_4 are in the room, and u_1 change the room condition manually, we destroy edge (u_2, u_1), so that u_1 and u_2 would not have the loop, which can be considered a conflict in hierarchy.

```
{
  "ucode": "00001C0000000000000020000000D4554",
  "name": "AC-RoomA304-1",
  "type": "air conditioner",
  "power": 1,
  "set_temp": 25.0,
  "room_temp": 25.796875,
  "fan_speed": 0,
  "fan_direction": 7
}
```


Figure 5.7: JSON Representation of Air Conditioner State

5.4 System Design & Implementation

The system configuration is shown in Fig. 5.6 depicts the smart building system in our architecture. We developed Smart Building Application Program Interface (API) which can be used to access devices such as sensors, actuators, air conditioner, lighting, and smart meter. We use Representational State Transfer (REST) API and choose JavaScript Object Notation (JSON) as the data format in our Smart Building API. Fig. 5.7 shows the JSON representation of air conditioner state. The occupants of the building can easily develop an application using Smart Building API. We use Ubiquitous ID (uID) [25] architecture, and assign a 128bit unique id to devices in the building.

We developed several agents for providing services to the occupants. *Room Agent* is responsible for providing service to the occupants of the room. When the occupant enter or leave the room, *Room Agent* will update the current occupant on the system and ask *Comfort Condition Agent* to provide an appropriate condition for the remaining occupants. *Device Agent* manages the device when an occupant manually changes device settings. *Comfort Condition Agent* is responsible for providing comfort condition to the occupants in the room based on proposed

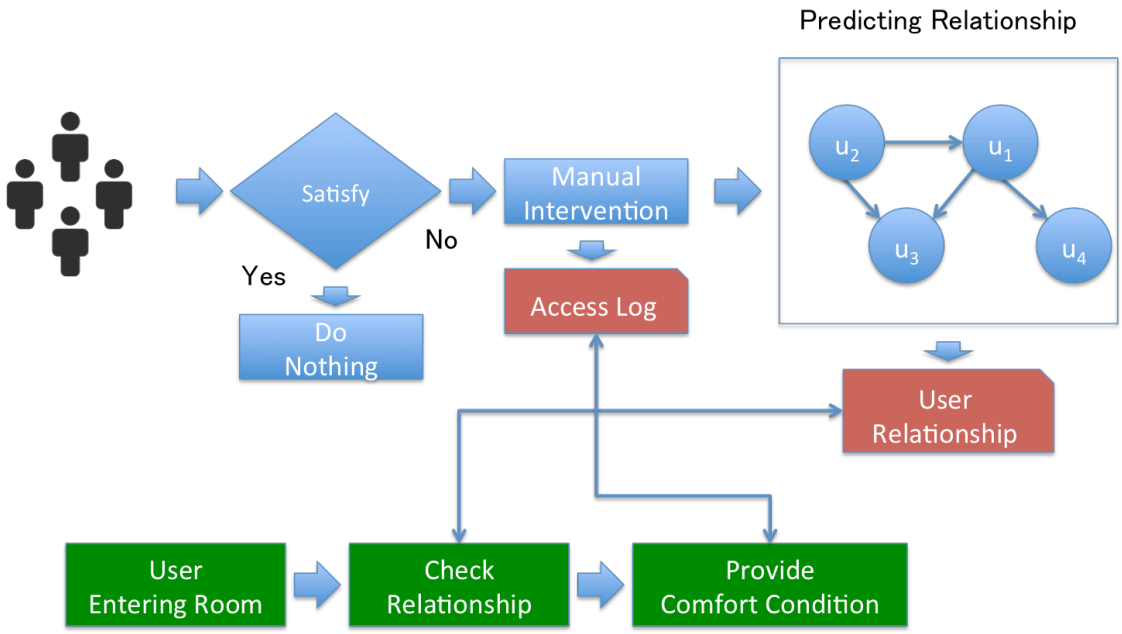


Figure 5.8: Flow of Service

algorithm. When the occupants change device settings, *Device Agent* will notice *Comfort Condition Agent* to update user preference of the related users. The occupants interact with the system through room control application which is explained in the next section. The access log includes user id, device id, device settings and timestamp is recorded in the database.

The Fig. 5.8 shows the flow of service in our implementation. How the proposed algorithm predicting the collective user preference is shown by the top part. After entering a room, the occupant will check if the room conditions are satisfied his comfort conditions or not. If he satisfies with the current conditions, he will do nothing. Otherwise, if he does not satisfy the current condition, he will do manual interventions to the appliance in order to change the current conditions. Every manual intervention to the appliance is recorded in the access log. Then, the system also changes the relationship among the current occupants. The bottom part shows how the agents providing service to the occupants.

5.5 Evaluation

In order to evaluate the proposed algorithm, we have developed a room control application in which the proposed algorithm is implemented. We evaluated the algorithm in the smart building named "Daiwa Ubiquitous Computing Research Building", located in Hongo campus of the University of Tokyo. We measured the number of manual interventions before and after the deployment of the proposed algorithm to the system. The manual intervention is done by an occupant when the current conditions of the room do not satisfy his comfort conditions. Therefore we are able to know comfort level of the occupant from the number of manual interventions.

5.5.1 Room Control Application

In order to let occupants interact with the smart building system, we have developed room control application. The room control application is web-based, the occupants interact and control a device in the room from the browser. Room control application also shows the current settings of the devices in the room. Therefore, the occupants are able to know the current settings before changing it.

5.5.2 Experiment Scenario

We evaluate the proposed algorithm in room A304 of Daiwa Ubiquitous Computing Research Building. The room is allocated to 4 occupants: one female and three males. We asked them to use the room control application when they want to change device settings. We used the lights and air conditioner as the devices in our experiment. We conducted the experiment into two phases.

First Phase, before deploying the algorithm

We recorded when and who enter and leave the room. We also recorded when and who access the device through room application and change the device settings.

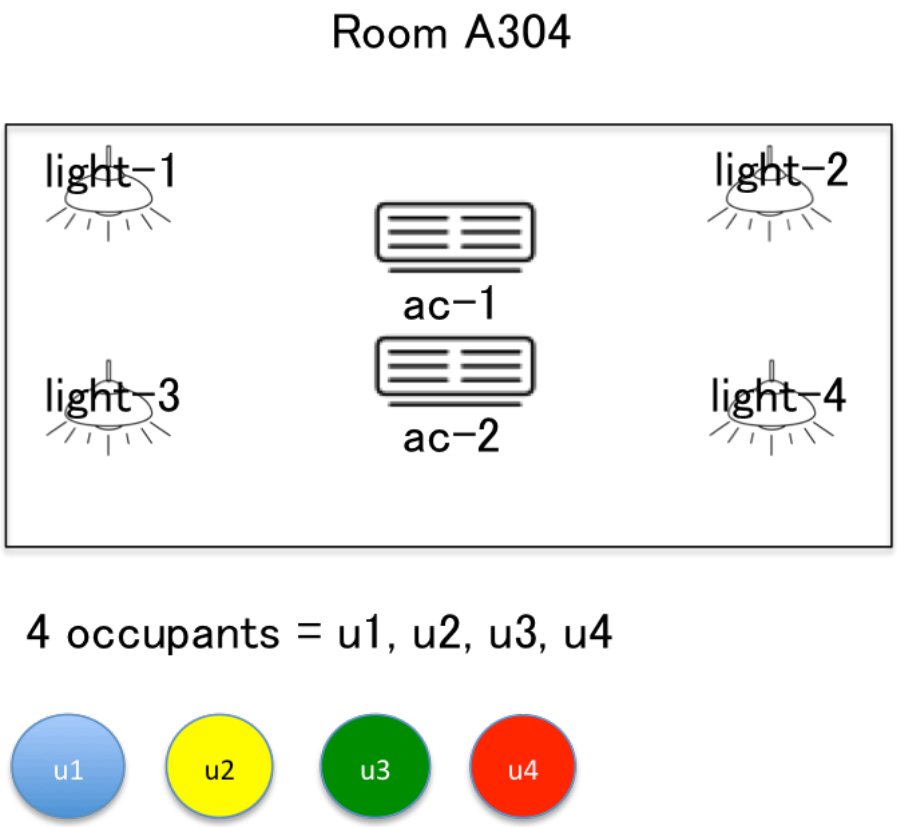


Figure 5.9: Room A304 of Daiwa Ubiquitous Computing Research Building

We conducted the first phase of the experiment for a period of one week. From the data of the first phase of the experiment, we obtained the user preference of all occupants. As shown in in Fig. 5.9, room A304 has four lights and two air conditioners.The user preference that we obtained include the temperature settings of the air conditioner 1, and the air conditioner 2, and which light that occupant usually use.

Second Phase, after deploying the algorithm

We use record of room state and access log that we obtained in the first phase of the experiment as the input of the proposed algorithm. As the output, we

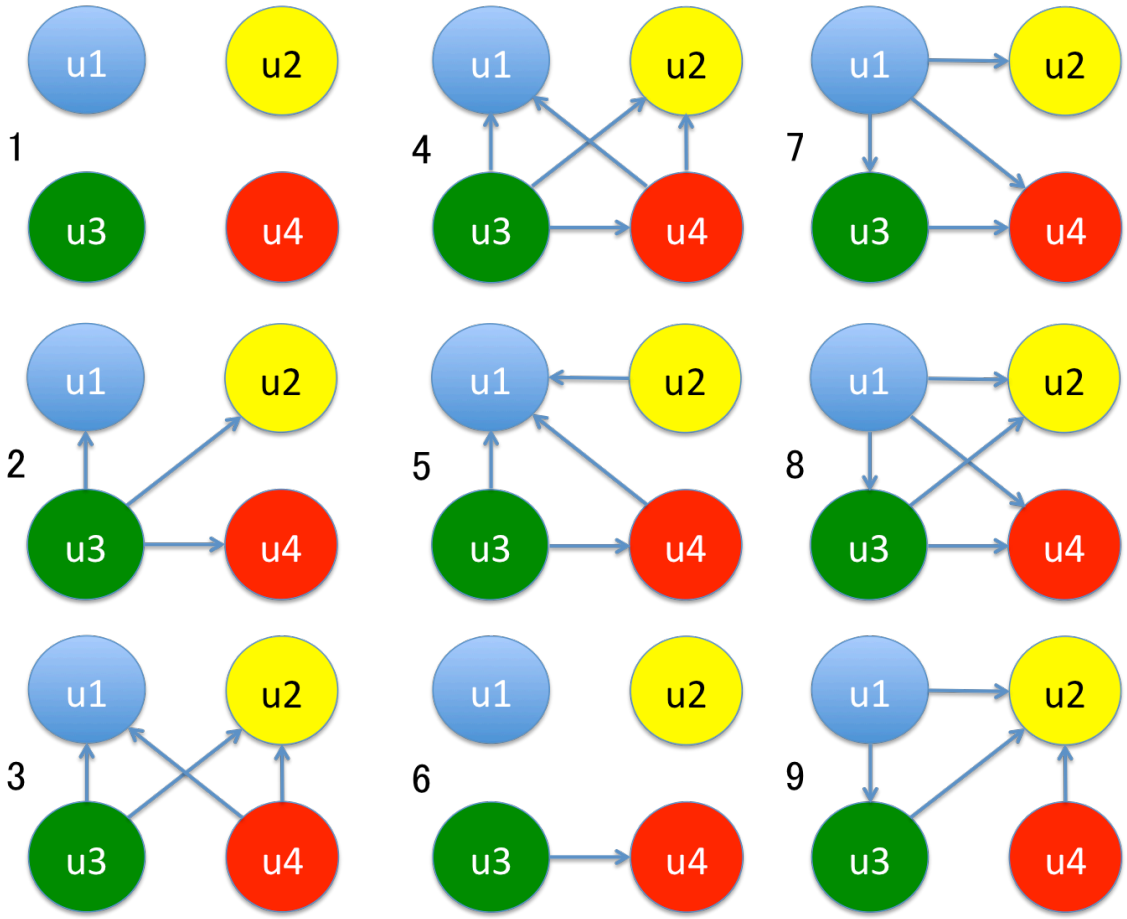


Figure 5.10: Transition of Relationship of Occupants

obtain the hierarchy of the occupants in room A304. We also predict the collective user preference of the occupants in room A304. We deploy the prediction to the system so that the system will automatically change the room conditions when the room state is changed. If the condition which is provided by the system do not satisfy the occupants, they can manually change the condition through room control application. We recorded the manual interventions in the second phase of the experiment for a period of one week.

5.5.3 Experimental Results

From the first phase of the experiment, we obtained the relationship of occupants in room A304. Fig. 5.10 presents the transition of the relationship of the occupants. In the initial stage, there are no edges in the graph. When one occupant accesses and changes the device settings, edges are created from the current occupant to all other occupants. In order to avoid inconsistency in the directed graph, we do not create the opposite direction of the edge. As shown in Fig. 5.10, after the occupants u3 changes the temperature of the air conditioner, the relationship changes from 1 to 2. Then the occupants u4 changes the temperature of the air conditioner, the relationship changes from 2 to 3. From this figure, it can be seen that proposed algorithm destroy the edge from u3 to u4 in order to avoid the inconsistency in the directed graph.

We show the comparison of the number of manual interventions in the first phase and the second phase of the experiment in Fig. 5.11. In the first phase of the experiment, we obtained 67 manual interventions from all occupants, which decreased to 24 manual interventions in the second phase of the experiment. The total number of manual interventions decreased by 64.2%. For each occupant, the number of manual interventions decreased as follows:

- for occupant u1, 80% (from 25 to 5)
- for occupant u2, 70% (from 10 to 3)
- for occupant u3, 60% (from 10 to 4)
- and for occupant u4, 45.5% (from 22 to 12)

The occupant u4 showed the least reduction of manual interventions. From the record of room state, we obtain that the occupant u4 was the first one enter room A304 most of the time. Since we did the experiment during the summer season, the first occupant who enter a room may set the temperature low than their comfort temperature in order to cool the room faster. Therefore, when the occupant u4

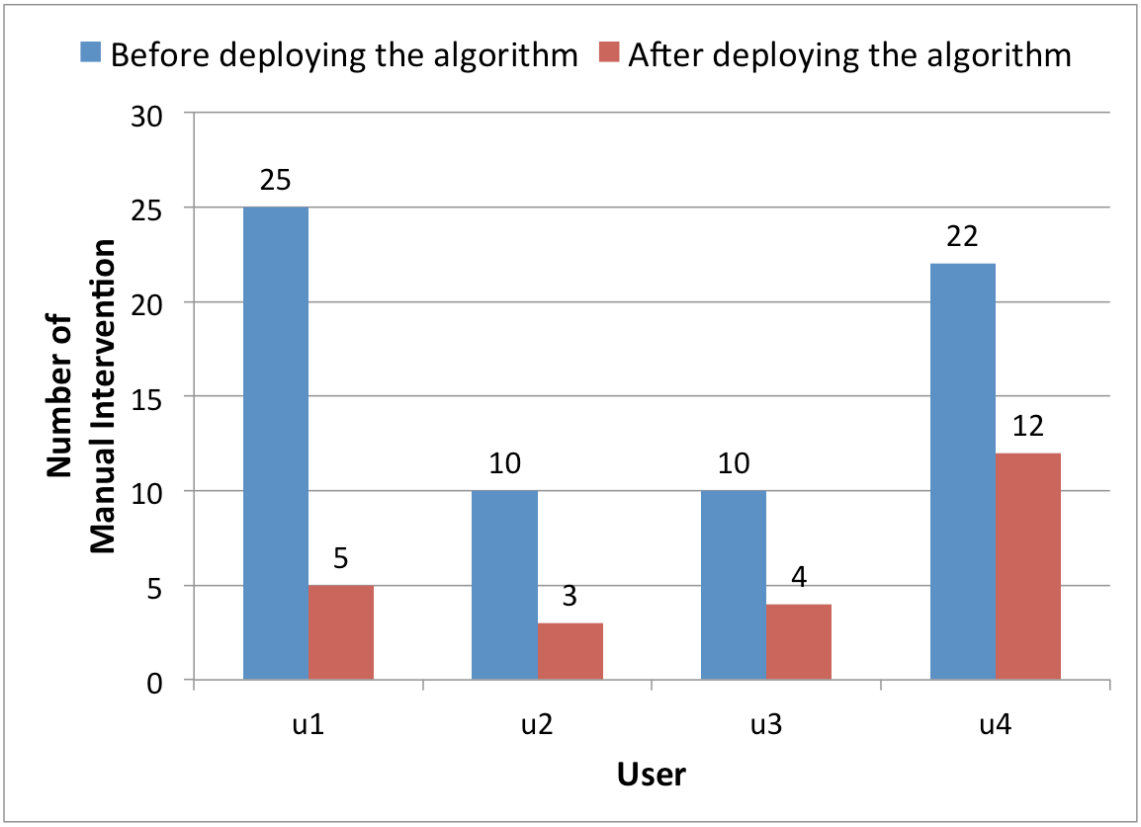


Figure 5.11: Comparison of The Number of Manual Intervention

enter room A304, he is not satisfied with the conditions provided by the system, and change the temperature lower than his comfort temperature. After the room become cool, he changes the temperature to his comfort temperature. Since the system records the last applied condition as the user preference, this case may occur for the user who enters the room for the first time.

This result shows that our proposed algorithm can automatically predict collective user preference from the user preference and the relationship of occupants. Since the total of the number of manual intervention decreased by 64.2% after the deployment of the proposed algorithm, our algorithm is feasible to optimize comfort level in smart buildings.

5.6 Summary

In this chapter, we have observed how the characteristic of organizational hierarchy of occupants in smart building influences a decision of comfort conditions for a group of occupants. In order to provide an optimal comfort level to the occupants, we have proposed an algorithm to predict collective user preference, unlike most other work which focuses on the single-user situation. The proposed algorithm is intuitive because it predicts collective user preference based on real life relationship of the occupants in smart buildings. We evaluated the proposed algorithm by measuring the number of manual intervention before and after deployment of the algorithm. The results show that the proposed algorithm reduces the number of manual intervention for all occupants. We believe that our proposed algorithm is feasible to optimize comfort level in smart buildings by predicting collective user preference.

Chapter 6

Discussion

In this chapter, firstly we discuss how the proposals solve the issue listed in chapter 1. Secondly, we discuss how the combination of our proposals becomes an IoT architecture, and discuss how the proposed architecture are able to realize the open services in the smart buildings.

6.1 De-Facto Standard API Design

In order to enable the combination and the utilization of heterogeneous IoT devices, we have proposed the de-facto standard of web API design, which is the combination between the device abstraction and uID architecture. The proposed standard API have implemented by using the HTTP protocol and the standard method such as GET, PUT, and POST. By providing device API as web API, we allow the developers to create applications independently. The development process does not depend on a particular programming language or programming framework. As a result, the developers are able to choose the language or framework which they know well.

To realize the de-facto standard API design, we have simplified the property of IoT devices by performing device abstraction. We have investigated a plethora of IoT devices which have different function, and found that they have two kind of properties: static property and dynamic property. We decided to perform device

abstraction based on these properties because the developer consider using two properties in different condition. Static property such as name, place, functionality, etc is describing the devices. The developer may use these properties for giving device description. Conversely, dynamic property is used to monitor and control the state of devices. As a result, it is easy to developers for memorizing the abstraction and utilizing the device maximally.

To manage the semantic knowledge of the device, we also used uID architecture in the proposed API design. We have assigned a *ucode*, 128-bit unique id, to devices entire the building. Since ucodes are assured to be always unique, static, and never reused, we can manage the device easily even the network topology is changed. The semantic knowledge relating to the devices are kept in the uID database. By adopting the uID architecture, we can manage the information connecting to the devices externally. As a result, the management of device can be performed in a flexible manner. For example, if the place of deployment is changed, the information relating the place can be changed without modifying the device directly.

We also proposed the addition of functionality description as the static property. The functionality description is provided by adding the state that can be controlled by the user. For example, if users are able to control the power, the temperature, and the fan direction of an air conditioner, all of the states are provided in the static property. Each state also contains the parameters can be sent to the device. By providing the functionality description, we have allowed the developer to generate the menu for controlling the device automatically. Furthermore, by providing the parameters, we can decrease the mistake of parameter error.

The proposal design has been implemented in the real smart building. The evaluation has been performed by comparing the development process which uses the proposed API design and traditional REST API. The experiment also has been conducted to confirm how much application developing time can be shortened by the proposed API design. The experimental result has shown that the average of developing time decreased by 73.9 % by using the proposed API design. It is indicated that the proposed design is successful to reduce the development cost

and allow the developers to combine heterogeneous IoT devices and create new applications easily. Through the proposed design, we have provided a new approach to become a reference for API designer who will design API for IoT devices or edge of IoT architecture.

6.2 Access Control Framework

In order to control access to a diversity IoT devices, we have proposed a novel access control framework. The proposed framework are able to handle resource-constrained device by delegating the complex security computation to the security manager. The proposed framework accommodate the dynamic environment by deploying the security manager as the third party which is not placed between the resource and the user. The proposed framework have supported the scalability and efficiency by splitting the security manager based on their function: authentication manager and access control manager.

To provide an easy deployment and extension, we have designed the authentication manager as REST API. To handle a various authentication method, we provide a parameter to describe what authentication method are used. By providing this parameter, we allow the system to specify which authentication method are required to access a device. We also utilized the concept of confidence level to describe the level of trust for one authentication method. By adopting the confidence level, we are able to define different critical level for requesting a device. Furthermore, we also have designed the access token to include the useful information which can be used in the authorization process. To protect the legitimacy of the token, we have attached a digital signature to the token which can be verified by the public key of authentication manager.

To simplify the deployment of access control process on the resources side, we have designed the access control manager in REST API. Since we designed a user to send a request directly to resources, we have to modify the resources to forward the request to access control manager. To provide an easy management of access to

device, we have adopted the policy-based access control. We have chosen XACML as the access control mechanism because it support a wide variety of data types, functions, and rules about combining the results of different policies.

The proposed framework has been implemented to support the open services in the real smart building. We have described that the proposed framework are able to deal with the threats in smart buildings. We also have explained that the framework is easy to use and supporting the scalability. The performance has been evaluated by conducting several scenarios of experiments. The experimental results have shown that the average of response time less than 0.1 seconds. It is indicated that the proposed framework is feasible to be practically used for controlling access to a diversity IoT devices in real smart buildings.

6.3 Services in Smart Buildings

To support the realization of ideas into services, we have described a concrete example of delivering services in smart buildings. To clarify what services are needed in a building, we have investigated the purpose of smart buildings, and classify services in the building which proposed in the prior works. Since the smart building environment is occupied by a group of users, we have chosen the service for providing comfort for a group of users as a concrete example.

To understand how the occupants reach their own comfort conditions, we have observed the behavior of the occupants. From the observation, we have found that the occupants will take an action only if the current conditions do not satisfy their comfort conditions. We also noticed that the comfort condition for a group of occupants can be known from the recent manual intervention on changing conditions.

In order to realize the comfort conditions, which is one of the purposes of the smart buildings, we have proposed an algorithm to predict a preference of collective users. We have described how the algorithm predicts the preference of a group of users. We also showed how the algorithm can be implemented by using Smart Building API which provided on the developer site.

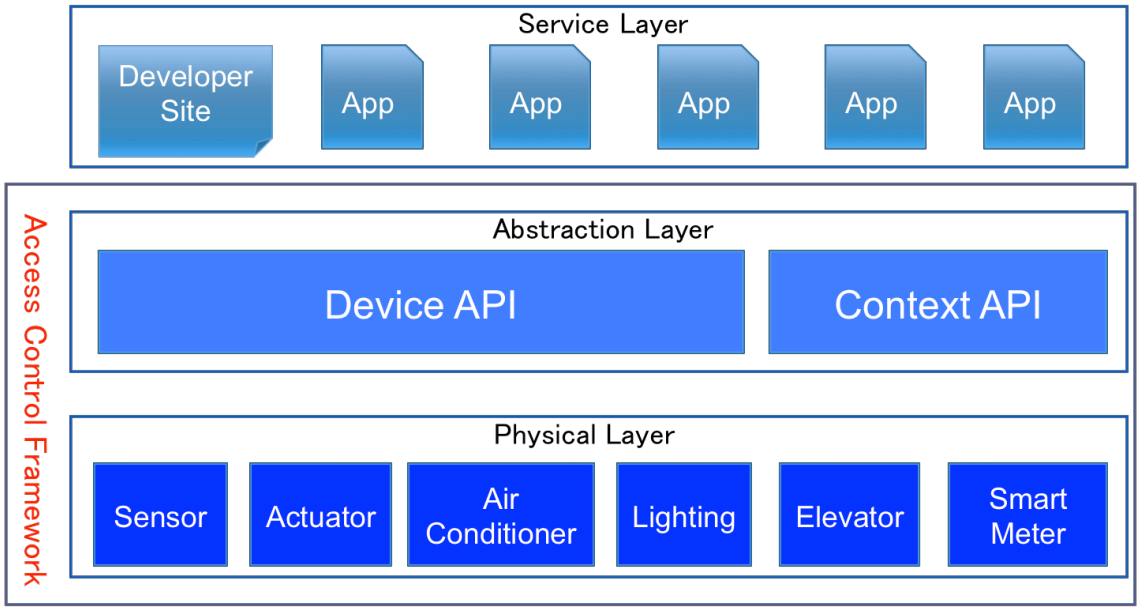


Figure 6.1: IoT Architecture

In order to evaluate the feasibility of the proposed algorithm, we have conducted an experiment by asking the occupants of one room to use the services. The experimental results showed that the number of manual interventions is decreased by 64.2 % by using the proposed algorithm. It is indicated that the proposed algorithm is feasible to become a service for optimizing comfort level of occupants. This also showed that our architecture is successful to support the realization of ideas into services.

6.4 IoT Architecture

The Fig. 6.1 shows IoT architecture which is the combination of the proposal in this thesis, for realizing open services in smart buildings. The architecture composed four main components. The first component is the physical layer, which contains a various network-enabled device. It may be sensors, actuator, home appliances, and etc. The second component is the abstraction layer, which contains

device APIs and context APIs. Since the physical layer consists of heterogeneous devices which have a different function, configuration, and specification, the abstraction layer is required to connect the physical layer and the service layer. The third component is the service layer, which contains application and developer site to deliver service to users. The documentation of abstraction layer, device APIs, and context APIs are published on the developer site. This allows the third developer to create new applications for delivering services to users. The fourth component is the access control framework. Access control framework is required to ensure that the authorized user only can access and control devices in the physical layer. By adopting the proposed architecture, we have shown that the open services can be realized in the smart buildings.

Chapter 7

Conclusion

In this research, we presented IoT architecture, a solution for providing open services by combining various IoT devices in smart buildings. Firstly, we proposed an alternative approach to standardize API design for heterogeneous devices in buildings environment. To handle heterogeneity of devices, we introduced device abstraction by investigating the property of devices. To minimize manual procedure on developing applications, we extended RESTful style by adopting a functional description. The second, we presented an access control framework for ensuring only the authorized user can access and control the devices. Lastly, we introduced a concrete example of delivering services in smart buildings. We developed a service for predicting collective user preference.

7.1 Summary

In this section, we present a summary of each chapter.

7.1.1 Related Work

In chapter 2, we discussed the previous work related to IoT architecture, security framework, and services in smart buildings. Relating to IoT architecture, we summarized a brief history of "Internet of Things", and IoT-enabling technologies. We also described the prior work which addressed a diverse, complex, and hetero-

geneous environment of IoT. However, none of the previous work has addressed properly how to standardize API design, which can reduce the cost of application development.

Regarding security framework, we discussed the conventional frameworks for enforcing security to resources. Further, we summarized prior work on enforcing security in the smart environment, which can be divided into two categories: model and implementation. Research on security model has focused on how to involve context to security enforcement, whereas research on implementation has focused on combining the existing framework. However, how to restrict access for heterogeneous IoT devices are still lacking.

In respect of services in smart buildings, we described a brief history of intelligent and smart buildings and services needed in the buildings. We summarized the prior work and classify it into two categories: services for reducing energy consumption and services for the optimizing comfort of occupants. In the summary section, we discussed the issues which did not address in the previous works. However, the previous works mainly focused on how to reduce the energy consumption, and investigation on how to improve the comfort level of occupants still not address properly.

7.1.2 De-Facto Standard API Design

In chapter 3, we proposed an alternative approach to standardize API design for open services in smart buildings. Firstly, we describe the lack of API standardization in the current IoT architecture, which leads to high cost of application development, which obstructs innovative application to emerging. Moreover, RESTful API, which is widely adopted in web services, lacks in functionality description.

In order to standardize API, we introduced the abstraction of devices based on the static property and the dynamic property. To overcome the lack of functionality description, we explored the useful information for describing devices in the development process and included it in the static property. To provide services, we

also presented context APIs, which is designed based on the relationship of three entities in smart buildings: space, occupant, and device.

In order to evaluate the feasibility of the proposed design, we implemented Smart Building API in the real smart building environment. As a use case, we developed Smart Room Application and compared the development process between traditional RESTful style and the proposed design. The results showed that the proposed design has been successfully reducing the development cost, and allowed the developers to create applications more easily.

7.1.3 Access Control Framework

In chapter 4, we presented a framework for restricting access to heterogeneous IoT devices in smart buildings. Since IoT environment such as smart building consists of a variety of devices including those with limited resource, the conventional access control framework from the cyber world cannot simply be adopted. Moreover, the management of appliances and devices in smart buildings is dynamic because they may be attached and removed any time depend on the necessity.

To deal with the heterogeneity of devices and dynamic environment in smart buildings, we proposed a framework which access restriction is performed by the security manager which is deployed as a trusted third party. By delegating complex security computation to the security manager, it enables the system to handle resource-constrained devices securely. To support scalability and efficiency, the security manager is split into authentication manager and access control manager. Authentication manager has a responsibility to verify the user and issue an access token for the verified user. The device is enforced to ask access control manager to evaluate all invocations received. The access control manager evaluate the invocations based on the access control policies which are composed by the administrator in advance.

To investigate the feasibility of the proposed framework, we implemented the framework in our smart building systems. Our analysis results indicated that

the proposed framework is adequate to restrict heterogeneous devices in smart buildings. We also conducted some experiments to evaluate the performance of the framework in the real smart building which contains more than 300 devices including sensors and appliances. In the experiments, we set the users to send a request for accessing the device, and we measured the response time of each request. We also measured the execution time on the server side. Experimental results showed that the proposed framework is feasible to be practically used in smart buildings.

7.1.4 Services in Smart Buildings

In chapter 5, we introduced the concrete example of services in smart buildings by combining IoT devices. Firstly, we described the purpose of smart buildings based on the investigation of the prior works. We found two main purposes of smart buildings: to reduce energy consumption and to provide comfort environment. As the concrete example of services, we observed the latter purpose, providing comfort environment. We found that how to provide the comfort environment in the building is challenging because the building spaces are occupied by a group of occupants. Therefore, we decided to investigate how to predict collective user preference in smart buildings.

We proposed an algorithm for services on automatically predicting collective user preference in smart buildings. The algorithm was designed based on two information which can be obtained by smart building API: access log and room state. The comfort conditions for the group of occupants can be obtained from the last applied conditions, which can be identified by using access log. The relationship of the occupants is generated by monitoring the situation in the room, which can be recognized by utilizing room state. The collective user preference can be predicted from the relationship of the occupants that have been generated and the comfort conditions that have been obtained.

We evaluated the proposed services by conducting several experiments in the

real smart building environment. We compared the number of manual intervention before and after the deployment of the proposed service. Experimental results showed that the manual interventions after the deployment of the proposed services decreased by 64.2 %, which indicated that the proposed services are feasible to provide comfort environment in smart buildings.

7.1.5 Discussion

In chapter 6, we have discussed how the proposal in chapter 3, 4, and 5 solve the issue listed in chapter 1. We also described our approach to addressing the technical problem and explained our finding during carry out this work. We have combined the proposals as the IoT architecture to realize open services in smart buildings.

7.2 Future Work

- In this thesis, we have proposed de-facto standard API design by performing device abstraction. To control the device, the knowledge regarding the device, for example, the id of the device is still required. It is important to investigate how to allow a user to control the environment without a requirement of device information. As the future directions, we consider observing *Service Level API*, in which users are able to control an environment without a requirement to know details of the device in the environment.

- In this work, we have implemented the proposed API design by using the HTTP protocol. Therefore, the third developer can choose freely the language or framework which they know well to create a new service. For people who have no programming experience, it is still difficult to realize their ideas into services. As the future work, we recommend investigating a new programming framework to develop applications easily.

- In access control framework, we use XACML as the access control mechanism.

The administrator has to define the policy in XML-based language and store it in the policy repository. To define the policy, the administrator is required to understand XACML. It is necessary to observe the policy composition for a non-technical administrator.

- As a concrete example, we have proposed a service for predicting collective user preference. Since one of the purposes of smart buildings is to reduce energy consumption, as the future directions, we consider developing service for efficient energy consumption.

References

[1] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[2] Dave Evans. The internet of things. *How the Next Evolution of the Internet is Changing Everything, Whitepaper, Cisco Internet Business Solutions Group (IBSG)*, 1:1–12, 2011.

[3] Gartner says 6.4 billion connected "things" will be in use in 2016, up 30 percent from 2015. `http://www.gartner.com/newsroom/id/3165317`.

[4] More than 30 billion devices will wirelessly connect to the internet of everything in 2020. `https://www.abiresearch.com/press/more-than-30-billion-devices-will-wirelessly-conne/`.

[5] Antonio J Jara, Miguel A Zamora-Izquierdo, and Antonio F Skarmeta. Interconnection framework for mhealth and remote monitoring based on the internet of things. *IEEE Journal on Selected Areas in Communications*, 31(9):47–65, 2013.

[6] Fei Tao, Ying Cheng, Li Da Xu, Lin Zhang, and Bo Hu Li. Cciot-cmfg: cloud computing and internet of things-based cloud manufacturing service system. *IEEE Transactions on Industrial Informatics*, 10(2):1435–1442, 2014.

[7] Yifan Bo and Haiyan Wang. The application of cloud computing and the internet of things in agriculture and forestry. In *Proceedings of the 2011 Inter-*

national Joint Conference on Service Sciences (IJCSS), pages 168–172. IEEE, 2011.

[8] Cristina Elena Turcu, Vasile Gheorghiţă Găitan, and Corneliu Octavian Turcu. An internet of things-based distributed intelligent system with self-optimization for controlling traffic-light intersections. In *Proceedings of the 2012 International Conference on Applied and Theoretical Electricity (ICATE)*, pages 1–5. IEEE, 2012.

[9] LI Lu-yi1 Zheng Yan-lin. The application of the internet of things in education [j]. *Modern Educational Technology*, 2(005), 2010.

[10] Dimosthenis Kyriazis, Theodora Varvarigou, Daniel White, Andrea Rossi, and Joshua Cooper. Sustainable smart city iot applications: Heat and electricity management & eco-conscious cruise control for public transportation. In *Proceedings of the 2013 IEEE 14th International Symposium and Workshops on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–5. IEEE, 2013.

[11] Hans Schaffers, Nicos Komninos, Marc Pallot, Brigitte Trousse, Michael Nilsson, and Alvaro Oliveira. Smart cities and the future internet: Towards cooperation frameworks for open innovation. In *Proceedings of the The Future Internet Assembly*, pages 431–446. Springer, 2011.

[12] Markus Jung, Christian Reinisch, and Wolfgang Kastner. Integrating building automation systems and ipv6 in the internet of things. In *Proceedings of the 2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, pages 683–688. IEEE, 2012.

[13] Liu Yang, Haiyan Yan, and Joseph C Lam. Thermal comfort and building energy consumption implications–a review. *Applied Energy*, 115:164–173, 2014.

[14] Anastasios I Dounis and Christos Caraiscos. Advanced control systems engineering for energy and comfort management in a building environment—a review. *Renewable and Sustainable Energy Reviews*, 13(6):1246–1261, 2009.

[15] Johnny KW Wong, Heng Li, and SW Wang. Intelligent building research: a review. *Automation in construction*, 14(1):143–159, 2005.

[16] Number of apps available in leading app stores as of june 2016. `https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/`.

[17] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[18] Luca Panziera and Flavio De Paoli. A framework for self-descriptive restful services. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13 Companion, pages 1407–1414, New York, NY, USA, 2013. ACM.

[19] Kevin Ashton. That 'internet of things' thing. *RFiD Journal*, 22(7):97–114, 2009.

[20] Mark Weiser. The computer for the 21st century. *Scientific american*, 265(3):94–104, 1991.

[21] Ken Sakamura. The objectives of the tron project. In *TRON Project 1987 Open-Architecture Computer Systems*, pages 3–16. Springer, 1987.

[22] Xiaolin Jia, Quanyuan Feng, Taihua Fan, and Quanshui Lei. Rfid technology and its applications in internet of things (iot). In *Proceedings of the 2012 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet)*, pages 1282–1285. IEEE, 2012.

[23] Gustavo Ramirez Gonzalez, Mario Muñoz Organero, and Carlos Delgado Kloos. Early infrastructure of an internet of things in spaces for learning. In

Proceedings of the Eighth IEEE International Conference on Advanced Learning Technologies, 2008. ICALT'08., pages 381–383. IEEE, 2008.

[24] Mihai T Lazarescu. Design of a wsn platform for long-term environmental monitoring for iot applications. *IEEE Journal on emerging and selected topics in circuits and systems*, 3(1):45–54, 2013.

[25] Noboru Koshizuka and Ken Sakamura. Ubiquitous id: standards for ubiquitous computing and the internet of things. *IEEE Pervasive Computing*, 4(9):98–101, 2010.

[26] BK Tripathy, Deboleena Dutta, and Chido Tazivazvino. On the research and development of social internet of things. In *Internet of Things (IoT) in 5G Mobile Technologies*, pages 153–173. Springer, 2016.

[27] Grant Hernandez, Orlando Arias, Daniel Buentello, and Yier Jin. Smart nest thermostat: A smart spy in your home. *Black Hat USA*, 2014.

[28] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, 17(4):2347–2376, 2015.

[29] Anna N Kim, Fredrik Hekland, Stig Petersen, and Paula Doyle. When hart goes wireless: Understanding and implementing the wirelesshart standard. In *Proceedings of the 2008 IEEE International Conference on Emerging Technologies and Factory Automation*, pages 899–907. IEEE, 2008.

[30] ZigBee Specification. Zigbee alliance. *URL: http://www. zigbee. org*, 558, 2006.

[31] Nandakishore Kushalnagar, Gabriel Montenegro, and Christian Schumacher. Ipv6 over low-power wireless personal area networks (6lowpans): overview, assumptions, problem statement, and goals. Technical report, 2007.

[32] Stuart Cheshire and Marc Krochmal. Multicast dns. Technical report, 2013.

[33] Stuart Cheshire and Marc Krochmal. Dns-based service discovery. Technical report, 2013.

[34] Zach Shelby, Klaus Hartke, and Carsten Bormann. The constrained application protocol (coap). Technical report, 2014.

[35] Dave Locke. Mq telemetry transport (mqtt) v3. 1 protocol specification. *IBM developerWorks Technical Library], available at http://www. ibm. com/developerworks/webservices/library/ws-mqtt/index. html*, 2010.

[36] A Stanford-Clark and A Nipper. Mq telemetry transport. 2014.

[37] Peter Saint-Andre. Extensible messaging and presence protocol (xmpp): Core. 2011.

[38] LM Ericsson. More than 50 billion connected devices. *White Paper*, 2011.

[39] Dominique Guinard and Vlad Trifa. Towards the web of things: Web mashups for embedded devices. In *Proceedings of the WWW (International World Wide Web Conferences) Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009)*, page 15, 2009.

[40] Zach Shelby. Constrained restful environments (core) link format. 2012.

[41] Jaeho Kim and Jang-Won Lee. Openiot: An open service framework for the internet of things. In *Proceedings of the 2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 89–93. IEEE, 2014.

[42] Asma Elmangoush, Thomas Magedanz, Alexander Blotny, and Niklas Blum. Design of restful apis for m2m services. In *Proceedings of the 2012 16th International Conference on Intelligence in Next Generation Networks (ICIN)*, pages 50–56. IEEE, 2012.

[43] Yannis Charalabidis, Charalampos Alexopoulos, Vasiliki Diamantopoulou, and Aggeliki Androutsopoulou. An open data and open services repository

for supporting citizen-driven application development for governance. In *Proceedings of the 2016 49th Hawaii International Conference on System Sciences (HICSS)*, pages 2596–2604. IEEE, 2016.

[44] Martin Serrano, Hoan Nguyen M Quoc, Manfred Hauswirth, Wei Wang, Payam Barnaghi, and Philippe Cousin. Open services for iot cloud applications in the future internet. In *Proceedings of the 2013 IEEE 14th International Symposium and Workshops on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–6. IEEE, 2013.

[45] Enzo Mingozzi, Giacomo Tanganelli, Carlo Vallati, and V Di Gregorio. An open framework for accessing things as a service. In *Proceedings of the 2013 16th International Symposium on Wireless Personal Multimedia Communications (WPMC)*, pages 1–5. IEEE, 2013.

[46] Simon Mayer, Nadine Inhelder, Ruben Verborgh, Rik Van de Walle, and Friedemann Mattern. Configuration of smart environments made simple: Combining visual modeling with semantic metadata and reasoning. In *Proceedings of the 2014 International Conference on the Internet of Things (IOT)*, pages 61–66. IEEE, 2014.

[47] Charilaos Akasiadis, Grigorios Tzortzis, Evaggelos Spyrou, and Constantine Spyropoulos. Developing complex services in an iot ecosystem. In *Proceedings of the 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 52–56. IEEE, 2015.

[48] Takeshi Yashiro, Shinsuke Kobayashi, Noboru Koshizuka, and Ken Sakamura. An internet of things (iot) architecture for embedded appliances. In *Proceedings of the 2013 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*, pages 314–319. IEEE, 2013.

[49] Floris Van den Abeele, Jeroen Hoebeke, Ingrid Moerman, and Piet Demeester. Integration of heterogeneous devices and communication models via the cloud

in the constrained internet of things. *International Journal of Distributed Sensor Networks*, 2015:1, 2015.

[50] Ruben Verborgh, Thomas Steiner, Davy Van Deursen, Sam Coppens, Joaquim Gabarró Vallés, and Rik Van de Walle. Functional descriptions as the bridge between hypermedia apis and the semantic web. In *Proceedings of the Third International Workshop on RESTful Design*, WS-REST '12, pages 33–40, New York, NY, USA, 2012. ACM.

[51] TRON Forum. T-kernel 2.0. *Website. Projektseite: http://www.tron.org/.*

[52] Enrique Carrillo, Victor Benitez, Cereza Mendoza, and Jesus Pacheco. Iot framework for smart buildings with cloud computing. In *Proceedings of the 2015 IEEE First International Smart Cities Conference (ISC2)*, pages 1–6. IEEE, 2015.

[53] Lu Hou, Shaohang Zhao, Xing Li, Periklis Chatzimisios, and Kan Zheng. Design and implementation of application programming interface for internet of things cloud. *International Journal of Network Management*, 2016.

[54] Hua Wang, Yanchun Zhang, and Jinli Cao. Access control management for ubiquitous computing. *Future Generation Computer Systems*, 24(8):870–878, 2008.

[55] Devdatta Kulkarni and Anand Tripathi. Context-aware role-based access control in pervasive computing systems. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies*, SACMAT '08, pages 113–122, New York, NY, USA, 2008. ACM.

[56] Jalal Al-Muhtadi, Anand Ranganathan, Roy Campbell, and M Dennis Mickunas. A flexible, privacy-preserving authentication framework for ubiquitous computing environments. In *Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops, 2002.*, pages 771–776. IEEE, 2002.

[57] Se Won Oh and Hyeon Soo Kim. Study on access permission control for the web of things. In *Proceedings of the 2015 17th International Conference on Advanced Communication Technology (ICACT)*, pages 574–580. IEEE, 2015.

[58] Sumayah Al-Rabiaah and Jalal Al-Muhtadi. Consec: Context-aware security framework for smart spaces. In *Proceedings of the 2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, pages 580–584. IEEE, 2012.

[59] Ji Eun Kim, George Boulos, John Yackovich, Tassilo Barth, Christian Beckel, and Daniel Mosse. Seamless integration of heterogeneous devices and access control in smart homes. In *Proceedings of the 2012 8th International Conference on Intelligent Environments (IE)*, pages 206–213. IEEE, 2012.

[60] Ezedine Barka, Sujith Samuel Mathew, and Yacine Atif. Securing the web of things with role-based access control. In *Proceedings of the International Conference on Codes, Cryptology, and Information Security*, pages 14–26. Springer, 2015.

[61] Grant Ho, Derek Leung, Pratyush Mishra, Ashkan Hosseini, Dawn Song, and David Wagner. Smart locks: Lessons for securing commodity internet of things devices. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '16, pages 461–472, New York, NY, USA, 2016. ACM.

[62] Konstantinos Fysarakis, Charalampos Konstantourakis, Konstantinos Rantos, Charalampos Manifavas, and Ioannis Papaefstathiou. Wsacd-a usable access control framework for smart home devices. In *Information Security Theory and Practice*, pages 120–133. Springer, 2015.

[63] Simone Cirani, Marco Picone, Pietro Gonizzi, Luca Veltri, and Gianluigi Ferrari. Iot-oas: An oauth-based authorization service architecture for secure services in iot scenarios. *Sensors Journal, IEEE*, 15(2):1224–1234, 2015.

[64] M Widdington and J Harris. Intelligent skins. *Reed Educational and Professional Publishing*, 2002.

[65] AH Buckman, M Mayfield, and Stephen BM Beck. What is a smart building? *Smart and Sustainable Built Environment*, 3(2):92–109, 2014.

[66] Paul Davidsson and Magnus Boman. Saving energy and providing value added services in intelligent buildings: A mas approach. In *Agent Systems, Mobile Agents, and Applications*, pages 166–177. Springer, 2000.

[67] Han Chen, Paul Chou, Sastry Duri, Hui Lei, and Johnathan Reason. The design and implementation of a smart building control system. In *Proceedings of the IEEE International Conference on e-Business Engineering, 2009. ICEBE'09.*, pages 255–262. IEEE, 2009.

[68] Jinsung Byun and Sehyun Park. Development of a self-adapting intelligent system for building energy saving and context-aware smart services. *IEEE Transactions on Consumer Electronics*, 57(1):90–98, 2011.

[69] Jianli Pan, Raj Jain, Subharthi Paul, Tam Vu, Abusayeed Saifullah, and Mo Sha. An internet of things framework for smart energy in buildings: Designs, prototype, and experiments. *IEEE Internet of Things Journal*, 2(6):527–537, 2015.

[70] Farrokh Jazizadeh, Ali Ghahramani, Burcin Becerik-Gerber, Tatiana Kichkaylo, and Michael Orosz. Human-building interaction framework for personalized thermal comfort-driven systems in office buildings. *Journal of Computing in Civil Engineering*, 28(1):2–16, 2013.

[71] Annie Chen. Context-aware collaborative filtering system: Predicting the user' s preference in the ubiquitous computing environment. In *International Symposium on Location-and Context-Awareness*, pages 244–253. Springer, 2005.

[72] María V Moreno, Miguel A Zamora, and Antonio F Skarmeta. User-centric smart buildings for energy sustainable smart cities. *Transactions on Emerging Telecommunications Technologies*, 25(1):41–55, 2014.

[73] Anika Schumann, Nic Wilson, and Mateo Burillo. Learning user preferences to maximise occupant comfort in office buildings. In *Proceedings of the International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 681–690. Springer, 2010.

[74] Miguel A Zamora-Izquierdo, José Santa, and Antonio F Gómez-Skarmeta. An integral and networked home automation solution for indoor ambient intelligence. *IEEE Pervasive Computing*, 9(4):66–77, 2010.

[75] Der-Yeuan Yu, Ettore Ferranti, and Hadeli Hadeli. An intelligent building that listens to your needs. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 58–63, New York, NY, USA, 2013. ACM.

[76] Shinya Yamamoto, Naoya Kouyama, Keiichi Yasumoto, and Minoru Ito. Maximizing users comfort levels through user preference estimation in public smartspaces. In *Proceedings of the 2011 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pages 572–577. IEEE, 2011.

[77] Zhi-Hao Lin and Li-Chen Fu. Multi-user preference model and service provision in a smart home environment. In *Proceedings of the 2007 IEEE International Conference on Automation Science and Engineering*, pages 759–764. IEEE, 2007.

[78] Ken Sakamura. Programable architecture: A smart control system in daiwa ubiquitous computing research building, 2014.

[79] David Heinemeier Hansson et al. Ruby on rails. *Website. Projektseite: http://www. rubyonrails. org*, 2009.

[80] OASIS Standard. extensible access control markup language (xacml) version 3.0. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html, 2013.

[81] Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. pages 407–416, 2000.

[82] Michael Jones, Paul Tarjan, J Bradley, Yaron Goland, Nat Sakimura, John Panzer, and Dirk Balfanz. Json web token (jwt). 2012.

[83] Simon Josefsson. The base16, base32, and base64 data encodings. 2006.

[84] Balana engine. https://github.com/wso2/balana.

[85] Dick Hardt. The oauth 2.0 authorization framework. 2012.

[86] Jakob Nielsen. *Usability engineering*. Elsevier, 1994.

[87] Alan D Kalvin and Yaakov L Varol. On the generation of all topological sortings. *Journal of Algorithms*, 4(2):150–162, 1983.