Case Study: ECHONET Lite Applications based on Embedded Component Systems

JieYing Jiang[1,a)] Hiroshi Oyama[2] Hiroaki Nagashima[3] Takuya Azumi[1]

Abstract: Nowadays, various electrical devices have not only offered convenience to people, but also increased the complexity of developing various devices in smart homes. Additionally, the number and type of the pieces of household electrical equipment pose a significant challenge for developers. Therefore, one must improve the development efficiency of electrical equipment in smart homes and facilitate the subsequent maintenance work by developers. To that end, this study proposes a development method based on embedded components (i.e., component-based development) to develop the devices in smart homes. The method can reduce the complexity of development, improve the development efficiency, increase the scalability, and facilitate future function update and maintenance. We develop and control smart-home devices based using the component description language TOPPERS Embedded Component System (TECS). When expanding new components or functions, TECS can automatically generate template C files and implement the functions contained in them, thereby offering improved scalability, while reducing the complexity of future redevelopment.

Keywords: Component-based development, Smart home, Internet of Things, ECHONET Lite

1. Introduction

Recently, the rapid development of the Internet of Things (IoT) has led to the continuous expansion of the scale of embedded control system software. There are different lifestyles in different parts of the world, thereby resulting in different types and numbers of the pieces of electrical equipment in each home. This causes a huge problem for developers, and the number and types of electrical devices are a substantial development burden for developers. Additionally, the functions of the IoT systems have become more powerful than before, although the structure is becoming increasingly complicated. By 2022, there will be approximately 29 billion Internet-enabled connected objects [1] that will fall under the IoT label. Therefore, the requirements associated with the abilities of the developers of IoT components have also become higher, and the development time and costs have also increased. Because of the increasingly complex structures of huge embedded systems, it will become more inconvenient to maintain and develop the components in the future.

We, therefore, should propose a development method that developers find convenient to maintain and expand the embedded systems. Because assembly languages are not universal and modular, they are highly non-portable. Therefore, in embedded software applications, assembly languages should be used to the least possible extent, and using high-level languages with good portability for development can effectively improve the portability and reusability of application software programs. Therefore, it becomes important to find an effective development method to assist developers, improve the development efficiency and reusability, reduce the development costs, and promote future maintenance and expansion. Reducing the complexity of design is particularly important to ensure swift development and a correct software product. To this end, component-based software development approaches are particularly appropriate [2].

The component-based development (CBD) method emphasizes the separation of concerns with respect to the wide-ranging functionality available throughout a given software system. It is a reuse-based approach to defining, implementing, and composing loosely coupled independent components into systems. It is highly flexible, as developers can add or delete components as needed to quickly build the application software, thereby facilitating initial development and maintenance.

To make people's lives more convenient, with the support of smart homes, the electrical equipment in the home is developed to facilitate its easy usage. With the implementation of the integration technology, communication technology, interoperability and cabling standards, smart home networks will continue to improve. This includes the operation and management of all the smart furniture, equipment, and systems in the home network by the application of integrated technologies. The technical characteristics of a smart home are as follows. Through the home gateway and its system software, a smart home platform system is established. The home intelligent terminal uses the computer technology, microelectronic technology and communication technology to integrate all the functions of the home intelligence, and thus a smart home is built on an integrated platform. The external expansion components realize the connection and control of the household appliances. The development board and all the components are connected through an embedded system.

This study proposes a method for developing embedded com-

[1] Graduate School of Science and Engineering, Saitama University, Sakura-ku, Saitama 338–8570, Japan
[2] OKUMA Corporation
[3] Cores Co., Ltd.
[a)] jan.k.413@ms.saitama-u.ac.jp

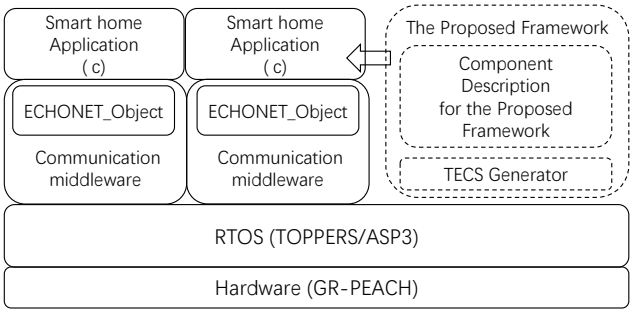


Fig. 1: System model of the proposed framework

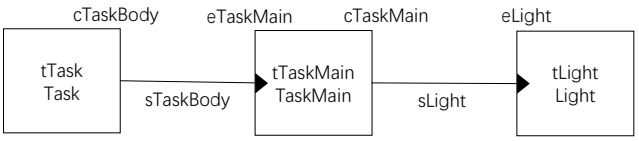


Fig. 2: Component diagram

```
1 signature sLight{
2   ER onOffPropertySet ([in,size_is (size)] uint8_t *src, [in]int size, [out]
        bool_t *anno);
3 };
```

Fig. 3: Signature description

ponents based on TOPPERS Embedded Component System (TECS). The method uses the Temple code generated by TECS generator to develop and extend the components in the smart home.

The contributions of this study are as follows.

- Improves the scalability of smart homes: The development framework of TECS for smart homes is appropriate for local smart homes. It can compile the device files, and then connect and control the smart home system through the embedded board. This will enhance the scalability of local smart home systems.
- Reduce the difficulty of redevelopment and maintenance: When updating and maintaining the functions of the existing equipment in smart homes, there is no need to perform complete compilation and debugging, thereby reducing the complexity of redevelopment.
- Easy to add and delete: The development method is highly configurable, and thus the device functions extended by the framework are easy to add and delete.

Organization: Section 2 introduces the system model and the basic technologies involved, including TECS. Section 3 describes the process and method of realization. Section 4 discusses the development methods of smart home development frameworks and components in related works. Section 5 introduces future work and research directions.

2. SYSTEM MODEL

This section describes the system model of the proposed framework (see **Fig. 1**). It contains two main parts. The first part introduces the basic information based on TECS and the basic development framework. The second part introduces ECHONET Lite.

2.1 TOPPERS Embedded Component System

TECS is a component system designed for developing embedded systems [3]. Its development method is to firstly divide the project into components and subsystems, and then develop applications into components by using subsystems. This method of the development can improve the reusability of the embedded software, thereby improving the production efficiency and reducing the development costs. TECS can be used in several domains of embedded systems because it supports multi-sized and diverse components [4].

TECS has statically developed components, which will not generate any excessive overhead when running, thereby effectively reducing the memory resource requirements. The control of a smart home and the detection of its status are performed in real-time, and thus we also develop components based on a real-time system. TECS can deal with real-time OS (RTOS) resources (such as tasks and semaphores) as components [5], and the TECS generator plugin can componentize RTOS features, thereby requiring complex static API generation using the proposed plugin [6]. Thus, TECS meets the requirements of smart home development. The TECS design is as follows.

(1) Component model: **Figure 2** shows an example of a component. It can help developers better understand the structure of the component. Each *cell* is an instance of a TECS component, and it includes *entry ports*, *call ports*, attributes, and variables. Each *celltype* defines the *entry ports*, *call ports*, attributes, and internal variables. The *entry port* is an interface that provides functions to other *cells*. The *call port* is an interface that uses the services of other *cells*. A *cell* communicates in the environment through these interfaces. The ports of each component are defined by its *signature* (i.e., sets of functions), which defines the component's interface.

(2) Component description: We use the component description language (CDL) to describe the components in TECS. In TECS, a component comprises a *signature*, *celltype*, and single *cell* description specified using CDL, as shown follows.

a) *Signature* description: Its function is to declare the name of the *signature* and its related functions. The keyword "*signature*" is followed by the *signature* name, which is prefixed with "s," e.g., sLight (see **Fig. 3**). The *signature* body includes a set of unit interface functions, each of which is described in the C language. The function parameters must contain specifiers for the input and output, such as "[in]" and "[out]."

b) *Celltype* description: It defines a component's *celltype*, including the component's *entry port*, *call port*, attributes and variables, as shown in **Fig. 4**. The *call port* is used to call the entry function of other units, and the *entry port* is used to provide functions. We use the keyword "entry" to define the *call port*, followed by the *signature* (e.g., sLight) and port name (e.g., cTaskMain). The definition of the inlet port is similar to that of the *call port*. The attributes and variables are defined using keywords "attr" and "var," followed by a name.

c) *Cell* description: It is used to instantiate and connect *cells*, as shown in **Fig. 5**. The keyword "*cell* " is followed by the *cell-*

```
celltype tTaskMain{
    entry sTaskBody eTaskMain;
    call sLight cTaskMain;
};
celltype tLight{
    entry sLight eLight;
    attr {
      uint8_t  propertyCode;
      ATR   propertyAttribute;
      uint8_t  propertySize;
      intptr_t extendedInformation;
    };
};
```

Fig. 4: Celltype description

```
celltype tMain{
  cell tTask Task{
  cTaskBody = Main.eMain;
  stackSize = 4096;
  priority = 11;
  attribute = C EXP( "TA_ACT" );
};
```

Fig. 5: Cell description

type e.g., tTask, and name, e.g., Task. The *cells* are linked by specifying the *call port*, caller name, and *entry port*, in the same order. For example, in Fig. 5, the eMain *entry port* of the Main *cell* is connected to the cTaskBody *call port* of the Task *cell*.

(3) *Namespace*: TECS uses *namespaces* to prevent name conflicts. The *namespace* is similar to other languages, such as C++. In the given *namespace* of TECS, one can define other *namespaces*, so that the naming becomes hierarchical. **Figure 6** shows an example of TECS component description. In this example, the *celltype* and *signature* are defined in the *namespace*. This *namespace* is nLight, followed by the keyword "*namespace*" with the prefix "n." We then define the required *celltype* and signed content in this *namespace*. *Namespace* identifiers are used to refer to *signatures* and *celltypes* from different *namespaces*.

(4) *Region*: We introduce the concept of *region* to control the layout of the unit. It has the function of preventing name conflicts between units including *namespaces*. A *region* is a meaningful unit in programs, and it can be used to separate the core structure and memory structure, from one another. We define a *region* by using the keyword "*region*," followed by the name of the *region*, prefixed with "r," as shown in Fig. 6. We then set the unit required in the *region*. The *region* identifier is used to refer to the *cells* in other *regions*.

(5) Development flow: **Figure 7** shows the development process when using TECS. Notable, the TECS development can be divided into two modules, namely, component design and application development. First, the TECS generator generates RTOS configuration files (*.cfg) from CDL files. We then define the *signature* and *celltype* through component design, and use the C template code (*celltype* code) generated by the TECS generator to implement the component functions. We use application diagrams and predefined *celltype* to develop applications. The application module is generated by connecting the header file, and compiling the interface code, and the *celltype* code.

2.2 ECHONET Lite

Energy conservation and homecare network (ECHONET) [7] has become a home network standard certified by ICE and ISO. ECHONET device objects not only provide simple interaction at the level of remote control commands (such as the "ON/OFF" button), but also standardize the functions of highly complex devices, thereby enabling the advanced control required by energy management applications [8]. However, the ECHONET protocol is not widely used. First, the ECHONET specification requires that the system configuration for multiple controllers and multiple devices to be fairly complex. Second, the overall complexity of the ECHONET protocol results in some compliance implementations. Therefore, the ECHONET protocol was redesigned as the substantially simplified ECHONET Lite protocol in 2011.

The ECHONET Lite network protocol provides a standard method of controlling the household appliances to achieve interoperability between devices that are from different manufacturers in the smart home. The development concept of ECHONET Lite is based on the ECHONET standard, but its structure can be easily tackled by home network system builders and service system developers [9]. Although devices that comply with the ECHONET Lite Specification and those that comply with the ECHONET Specification cannot be interconnected, they may coexist in the same system.

```
namespace nLight {
  celltype tTaskMain{
    entry sTaskBody eTaskMain;
    call sLight cTaskMain;
  };
  celltype tLight{
    entry sLight eLight;
  };
};
region rLight {
  cell tTask Task{
    cTaskBody = TaskMain.eTaskMain;
    priority = 1;
    stackSize = 81920;
    attribute = C_EXP("TA_ACT");
};
  cell tTaskMain TaskMain{
    cTaskMain = Light.eLight;
};
  cell tLight Light{
};
};
```

Fig. 6: Description of namespace and region

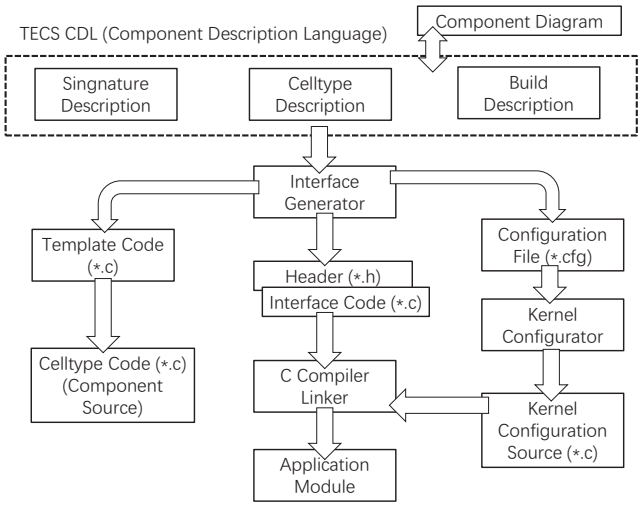


Fig. 7: Development flow

Table 1: Specifications of the board

Board	GR-PEACH
CPU	Cortex-A9 RZ/A1H 400MHz
Flash ROM	8MB
RAM	10MB
LAN Controller	Controller LAN87

ECHONET Lite has emerged as a leading interface used in smart homes in Japan, and accordingly the number of devices compatible with the ECHONET Lite protocol is steadily growing [7]. **Figure 8** shows the basic concept of ECHONET Lite. The network of ECHONET Lite node is a collection of devices. A node is a physical device connected to the network. Each node contains the network address, profile file objects that identify a node, and a list of device objects. Notably, the device objects provide a standardized method to represent the device resources and services through attribute lists and the constraints of each attribute. The latest ECHONET specification [10] released in March 2020 defines 116 different device objects.

Figure 9 shows the ECHONET Lite system architecture. The largest area that ECHONET Lite can manage is referred to as a domain. A domain is specified as the variety of controlled resources (e.g., home equipment, appliances and consumer electronics, sensors, controllers, and remote controls) present within the network range determined by ECHONET Lite [9]. A system performs communication and linked operations between devices and controllers, which monitor/control/operate the devices themselves. Because a domain might include one or more systems, the same device or controller can exist in more than one systems. When connecting a system to another system that lies outside the domain, an ECHONET Lite gateway is used as an interface.

The ECHONET Lite standard defines the communication between components called ECHONET objects. The ECHONET object has certain properties, and the actual device is operated on the basis of this property. This middleware provides 1) a static API that defines ECHONET objects, and 2) a static API that defines the properties. User-defined ECHONET objects and their properties in the static API are converted to C source codes in the TOPPERS kernel configurator, and the code is then compiled and linked with the application software.

3. DESIGN AND IMPLEMENTATION

This section compares some methods of smart home development with one another and introduces our development framework. It also introduces the software and hardware parameters involved.

Kadima et al. proposed hardware implementation of a multi-platform control system for home automation [11]. Such systems belong to a domain commonly referred to as smart home systems. By using a Raspberry Pi microcomputer to design IoT home automation system equipment, and by using an Arduino Uno board as the controller and relay to connect the bulb and circuit, in wireless communication, radio frequency waves are used to transmit signals, and in wired communication, wires are used to transmit signals. This method makes IoT easy to use and reduces the complexity via the ability of one device to control various other devices. Additionally, it combines hardware and software technologies and can be easily used in smart home automation applications.

Einarsson et al. proposed a SmartHomeML based modular language, which is a domain-specific modeling language for smart home applications [12]. SmartHomeML uses model-to-text conversion templates to generate smart home adaptation services that meet the target provider's specifications, thereby allowing users to define new skills, and utilize template-based model-to-text conversion to automatically generate adapters and connect to smart home devices. It can be used on other development platforms, such as the Amazon Alexa platform. Through the mapping relationship between the module language and development platform, SmartHomeML can be applied to smart home development, thereby reducing the difficulty of development, although it takes time to sort out the mapping relationship.

Our proposed development framework is based on the development of embedded components, and it can be extended to local smart home devices. Compared with the development methods of Kadima and Einarsson, our development process and structure are clearer. While extending the functions of any equipment that belongs to the ECHONET Lite protocol, we need not spend too much time to familiarize ourselves with the mapping relationship between the module language and development platform. We can create the *celltype* code by using the template code created by the generator, and then we can use the template code to implement the functions we need.

This section describes the design and method of development. GR-PEACH is a development board used to implement functions. The detailed specifications of the demo board are presented in **Table 1**. We connected the board to a host PC via a LAN cable and evaluated the data sending and receiving.

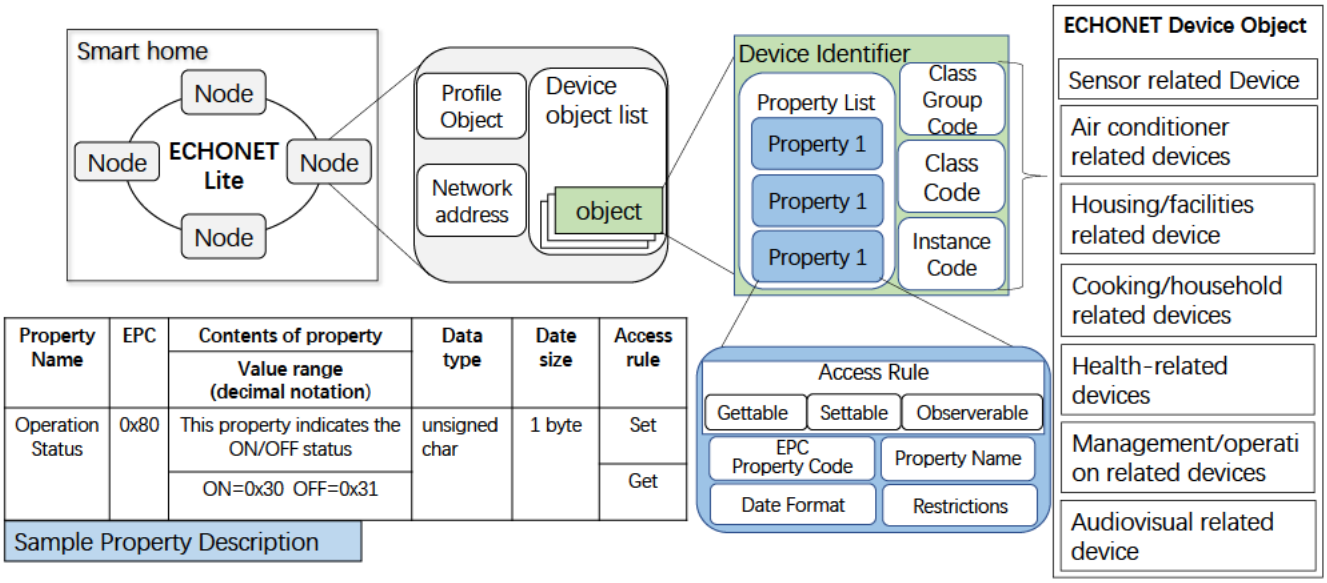
In this experiment, the development board was connected to the PC through the Ethernet, following which the PC could control the development board through the Ethernet. The PC need not be connected to Wi-Fi. Our example is to connect the GR-PEACH development board to a PC via LAN. The PC can then access and control the IP address of GR-PEACH through the super speed node generator (SSNG) for ECHONET Lite .

3.1 Super Speed Node Generator

The SSNG software was developed by Smart Home Research Center, Kanagawa Institute of Technology Engineering. As shown in **Fig. 10**, in the SSNG software, EHD stands for the ECHONET Lite header, TID the Transaction Id, SEOJ the Source ECHONET Lite object, DEOJ the destination ECHONET Lite object, ESV the ECHONET Lite service, OPC the Operation count, EPC the ECHONET Lite Property code, PDC the Property data count, and EDT the ECHONET Lite data.

In the example provided in this study, the goal is to control the Light connected to GR-PEACH. Therefore, in the SSNG software, SEOJ selects the controller, and DEOJ selects the general lighting class. The other parameters are maintained at their default states.

We can also install SSNG in Node.js. SSNG for Node.js is



Property Name	EPC	Contents of property Value range (decimal notation)	Data type	Date size	Access rule
Operation Status	0x80	This property indicates the ON/OFF status ON=0x30 OFF=0x31	unsigned char	1 byte	Set Get

Sample Property Description

Fig. 8: Concepts of the ECHONET Lite protocol

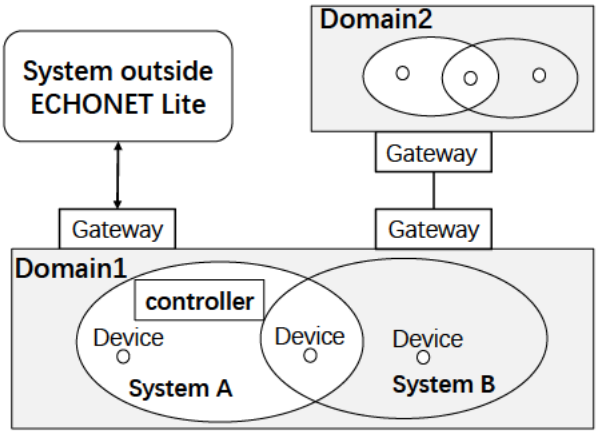


Fig. 9: ECHONET Lite system architecture

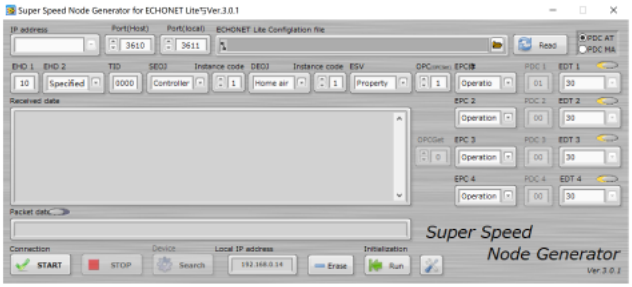


Fig. 10: SSNG software

a tool to send and receive the ECHONET Lite command. First, type data in the input fields of IP address, DEOJ, ESV, EPC and EDT. On clicking the SEND button, then the ECHONET Lite command is sent. The received ECHONET Lite data are automatically displayed in the packets monitor area. We enter SSNG in the command window, and then when the GR-PEACH development board is connected, following which it will prompt port bind ok. We can then control the embedded board by opening the localhost 3000 in the browser.

3.2 TERA TERM

Tera Term can provide communication between platform-independent web-based application servers and any remote Telnet/SSH host. We can build web applications that can control, monitor, or provide any device with telnet or SSH-based command interface.

We connected GR-PEACH to the computer through the Tera Term software and set the speed in the serial port setting to 115,200 bit/s. We then pressed the reset button on the embedded board, and the bin file information written in the embedded board was displayed in Tera Term.

3.3 TECS CODE

This case is based on the implementation of the ECHONET Lite project, and the application for controlling the embedded board LED lights in the project is implemented through TECS. In the light project, there exist two ways to control the components through Bluetooth and Ethernet, respectively. Because our embedded board did not have any Bluetooth module, we controlled the embedded board through Ethernet. At the beginning of application development, the development framework structure should be drawn, as shown in Fig. 2. In the development diagram, we need to define the *celltype* and *cell* of the main function and the *celltype* and *cell* of the component to be implemented, and then connect their *cells* through the *signature*. The TECS component diagram can be drawn before application development. The TECS component diagram can help us understand our development framework intuitively during the application development period.

In this example, two *celltypes* are defined in cdl file, as shown in **Fig. 11**. The first *celltype* is tTaskMain, and a TaskMain *cells* is defined in this *celltype*. The second *celltype* is tLight, and a Light *cell* is defined in this *celltype*. Both the *celltypes* are connected through the *signature*, and the function headers in the all called *cells* must be written into the *signature*. A *signature* named

```
signature sLight{
    ER onOffPropertySet ([in,size_is(size)]uint8_t *src, [in]int size,
    [out] bool_t *anno);
};
celltype tTaskMain{
    entry sTaskBody eTaskMain;
    call sLight cTaskMain;
};
cell tTaskMain TaskMain{
    cTaskMain = Light.eLight;
};
celltype tLight{
    entry sLight eLight;
    attr {
      uint8_t  propertyCode;
      ATR   propertyAttribute;
      uint8_t  propertySize;
      intptr_t extendedInformation;
    };
};
cell tLight Light{
    propertyCode = 0;
    propertyAttribute = 0;
    propertySize = 0;
    extendedInformation = 0;
};
```

Fig. 11: TECS code

sLight is defined between tTaskMain and tLight. There is a function header of the called *cell* in sLight.

In the *signature*, there will be a *call port* and an *entry port*. Each *cell* calls the functions in other *cells* through the *call port*. The *call port* defined in the TaskMain *cell* is cTaskMain, and the *entry port* is called eTaskMain. The *entry port* defined in the Light *cell* is named eLight. Functions are called only in the Light *cell*, and thus there is no need to set the *call port*, but just provide the *entry port* for other *cells* to call the functions.

We defined a function named onOffPropertySet to control the Light connected to GR-PEACH. This function can control the on and off of the lights. The onOffPropertySet function is essentially a switch to control the open and close state of the lights. When the input is 0x30, GPIO outputs 1 and the light turns on. However, when the input is 0x31, GPIO outputs 0 and the light is turned off, as shown in **Fig. 12**.

3.4 TECS GENERATOR

After defining the *celltype*, *cell* and *signature* in the cdl file, execute the make operation, and then the TECS generator will generate tTaskMain and tLight template files in the gen folder, as shown in Fig. 7. The generated C file contains several template codes. In the tTaskMain template code, all the header functions can call the functions in the generated file. We then use the TECSmerge which is used to create the required *celltype* codes for each *celltype*.

The TECSmerge command is mainly used in the following two situations.

- To create the first version of the *celltype* code based on the template code created by the generator.
- To reflect the changes in the CDL code to the existing *celltype* code.

A template C file is generated through the TECSmerge command, and its named after is the name of the *cell* we defined. All the header functions can call the functions in the file. We then call the required function in the main function of the C file of the *call port*. The required function is defined in the C file of the *entry port*. Then the file is compiled using the C compiler. After successful compilation, this function has been realized.

```
ER eLight_onOffPropertySet (CELLIDX idx, [in, size_is(size)] uint8_t
      *src, int size, bool_t* anno){
  ER  ercd = E_OK;
  CELLCB *p_cellcb;
  if (VALID_IDX(idx)) {
    p_cellcb = GET_CELLCB(idx);
  }
  else {
    return(E_ID);
  } /* end if VALID_IDX(idx) */
  gpio_t pow_led;
  gpio_t relay_sw;
  gpio_t sw1, sw2;
  if (size != 1)
    return 0;
  *anno = *((uint8_t *)ATTR_extendedInformation) != *((uint8_t *)src);
  switch (*(uint8_t *)src) {
    case 0x30:
      *((uint8_t *)ATTR_extendedInformation) = *((uint8_t *)src);
      gpio_write(&pow_led, 1);
      gpio_write(&relay_sw, 1);
      break;
    case 0x31:
      *((uint8_t *)ATTR_extendedInformation) = *((uint8_t *)src);
      gpio_write(&pow_led, 0);
      gpio_write(&relay_sw, 0);
      break;
    default:
      return 0;
  };
};
```

Fig. 12: onOffPropertySet code

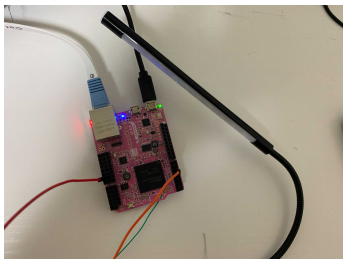
In this example, we need to generate a bin file to write to the embedded board. After writing the bin file to the embedded board, connect the embedded board to the computer via USB and LAN. Simultaneously, we need to use the Tera Term software. In Tera Term, select the serial port at which the embedded board is connected to the PC. Because the embedded board is connected to the computer via Ethernet, the PC needs to be disconnected from the Internet. Subsequently, find the IP address of the embedded board connected to the PC through the ipconfig command. Finally, enter the IP address of the embedded board in the SSNG application, and then we can control the embedded board through SSNG.

The final result is shown in **Fig. 13**. From Fig. 13 (a), it is evident that when EDT is 0x31, sending a command through SSNG to GR-PEACH, the light connected to GR-PEACH will turn off. In Fig. 13 (b), when EDT is 0x30, send a command through SSNG to GR-PEACH, and the light will turn on.

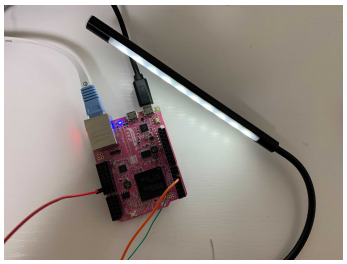
4. RELATED WORK

This section introduces the development and design of the mainstream smart home system models. There exist various smart home development platforms and methods, such as development through integrated modeling language [13], independent custom development [14], and design and implementation through open-source platforms [15]. **Table 2** compares the current component development methods with smart home development methods.

Shirata et al. proposed a component framework for obtaining the information about component systems [3]. The proposed framework supports obtaining static the information of the generated components and runtime component information during generation and execution periods. It uses a plugin to automati-



(a)



(b)

Fig. 13: The light application of ECHONET Lite on GR-PEACH

Table 2: Comparison among the related work

	EBD	ML	SC	CBD	RO
[4] [16] [5]	√	√		√	
[11] [17]	√		√	√	
[13] [15]		√	√	√	
[18]			√		
[2]	√			√	√
[19]	√			√	
[20]	√	√	√	√	
Our research	√	√	√	√	√

EBD : Embedded-based development

ML : Modeling language

SC : Scalability

CBD : Component-based development

RO : Reduce overhead

cally perform this operation, thereby automatically generating a unit that saves the information of the system. Plugin and dynamic connections can effectively reduce the number of lines of code, thereby improving the productivity and reusability. They are implemented as a set of components, so the developers can easily add or remove them in the project.

Yamamoto et al. presented an extended mruby on TECS framework for its application in developing software for IoT devices [20], including sensors and actuators. Each mruby program runs on a RiteVM mapped to a componentized task of an RTOS. Notably, mruby programs can call the TINET functions required for network software through the mruby-TECS bridge, and TINET+TECS can be applicable to various embedded systems. Thus, the software to be embedded in IoT devices can be developed. TINET+TECS for IoT devices improves the configurability and scalability and offers software developers high levels of productivity through variable network buffer sizes and also offers the ability to add or remove various TCP (or UDP) functions. Therefore, developers can easily add, remove, or reuse their functionalities as required.

Doan et al. described an IoTivity-based software architecture that is used to implement RES-Hub in a flexible and proposed the use of RES-Hub to ensure that the required functionalities were provided even when the cloud is unavailable [21]. During the normal functioning of the system, RES-Hub will receive regular status updates from the cloud. However, when the cloud is unavailable, RES-Hub uses the most recent state of the devices and services to continue the operation according to user specifications.

Guan et al. presented a novel integration test strategy (CREMTEG) for component-based real-time embedded software [16]. The strategy examines the states of all the components involved in a collaboration to exercise component interactions in the context of integration testing. This strategy solves the observability problems by recording the average value of each test pass, and it makes the difference between the expected and actual diagnosis results easily.

Banerjee et al. presented a centralized framework for managing the heterogeneous subsystems in a smart home. Additionally, they incorporated human-system interaction in this framework, by developing an Android application that received input in the form of motion gestures [22]. This framework allows multiple users to control different smart home appliances through their smartphones via a common platform. This framework is scalable, and new smart devices can be easily integrated into the smart home environment. A controller is the core entity of this framework, as it acts as a middleware between users and smart home appliances. It accepts user commands and translates them into device-specific instructions using their embedded APIs. Only users can access the smart appliances by using the instructions provided.

Einarsson et al. introduced SmartHomeML [12], a domain-specific modeling language for smart home applications, which allows users to define new functionalities. SmartHomeML comprises a model designer and a model generator, which use template-based transformation to automatically generate smart home device adapters and connectors for the target home control system.

Nandi et al. presented a statistical inference based approach for computing real-time contracts for component-based real-time control applications [19]. By verifying this system and checking the functionality and real-time properties, component-based real-time applications can be merged in an efficient and easy manner.

Steffen et al. proposed a component-based description language(CoDel) [17], which enabled system designers to use the parameterizable attributes of components, models, and interconnectivity to represent components as reusable building blocks of the system. CoDel can improve the usability, reusability, and scalability of the application components and models.

Khalilzad et al. proposed a framework that supported multi-resource, end-to-end resource reservation [2]. The framework utilizes a multiple-input multiple-output controller that coordinates the reserve size to track the dynamic resource requirements of the software components.

The SLASH framework [18] was proposed to design and implement an adaptive and self-learning smart home system. Big data and cloud computing technology were used as the main methods of analyzing the smart home data. Such a framework provides an effective model for designing the architecture and integration of smart home systems, thereby achieving scalability, maintainability, user-friendliness, and service combinations in an easy manner.

Mukendi et al. proposed the design and implementation of a customized IoT smart home [11]. By individually designing and implementing the functions of the smart home system, without considering devices with different regions or protocols, the complexity of the smart home system can be reduced. Mukendi used the Raspberry Pi microcomputer to design the IoT automation system, and used the Arduino development board as the controller to connect the circuits. The home automation system is equipped with a Wi-Fi module and LCD, which displays when the system is turned on or off. The user can control the on/off of various household appliances through a device, and can confirm the operating status of the device through the LED screen.

5. CONCLUSIONS

This study proposed a development method based on embedded components (i.e., CBD) to develop devices in smart homes. Through the embedded development technology, the software and hardware development of smart homes can be integrated in a quick and convenient manner. By developing CBD based smart homes, development modeling such as language modeling can be performed even before application development, and then the smart homes can be gradually developed using the established model diagram, thereby reducing the complexity of development. The development framework proposed is based on the development of embedded components to develop smart homes. It can reduce the complexity of development, improve the scalability of smart homes, and promote the update and maintenance of the device functions.

The purpose of this study was to develop components connected to the circuit board through CBD, and then control the components through the circuit board. Additionally, we aimed to improve the reusability and scalability of the components through CBD, and promote the update and maintenance of the component functions. In this case study, we have also developed USB watt meter, Light, Human detector, Air conditioning, Temperature sensor, Hot water pot, and Buzzer applications as components based on TECS.

On the basis of the current research, future research could improve the applicability of this development framework. Nowadays, it is becoming increasingly common to develop and control cloud-based smart homes, such as Amazon Alexa, and Google home. Our next research direction is to improve the adaptability of the framework to different development methods. Additionally, we wish to attempt at adapting the framework to various development environments, such as platform development and cloud development environments, to improve the scalability of smart homes.

References

[1] Gatouillat, A. and Badr, Y.: Verifiable and Resource-Aware Component Model for IoT Devices, in Proc. of Association for Computing Machinery, p. 235–242 (2017).

[2] Khalilzad, N., Ashjaei, M., Almeida, L., Behnam, M. and Nolte, T.: Adaptive multi-resource end-to-end reservations for component-based distributed real-time systems, *in Proc. of IEEE Symposium on Embedded Systems for Real-time Multimedia (ESTIMedia)*, pp. 1–10 (2015).

[3] Shirata, S., Oyama, H. and Azumi, T.: Runtime Component Information on Embedded Component Systems, *in Proc. of International Conference on Embedded and Ubiquitous Computing (EUC)*, pp. 166–173 (2018).

[4] Azumi, T., Yamamoto, M., Kominami, Y., Takagi, N., Oyama, H. and Takada, H.: A new specification of software components for embedded systems, *in Proc. of IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)*, IEEE, pp. 46–50 (2007).

[5] Azumi, T., Takada, H., Ukai, T. and Oyama, H.: Wheeled inverted pendulum with embedded component system: a case study, *in Proc. of IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, IEEE, pp. 151–155 (2010).

[6] Kawada, T., Azumi, T., Oyama, H. and Takada, H.: Componentizing an Operating System Feature Using a TECS Plugin, *in Proc. of the 4th IEEE International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA)*, IEEE, pp. 95–99 (2016).

[7] Pham, C., Makino, Y., Lim, Y. and Tan, Y.: Semantic Service Gateway for ECHONET based Smart Homes, *in Proc. of Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, pp. 175–179 (2019).

[8] Kodama, H.: The ECHONET Lite specifications and the work of the ECHONET consortium, *in Proc. of New Breeze-Quarterly of the ITU Association of Japan*, Vol. 27, No. 2, pp. 4–7 (2015).

[9] ECHONET: ECHONET-Lite Ver.1.13(01) E, `https://echonet.jp/spec_v113_lite_en/`.

[10] ECHONET: APPENDIX Detailed Requirements for ECHONET Device objects, `https://echonet.jp/spec_object_rm_en/`.

[11] Kadima, M. N. and Jafari, F.: A Customized Design of Smart Home Using Internet-of-Things, *in Proc. of the 9th International Conference on Information Management and Engineering*, ICIME 2017, in Proc. of Association for Computing Machinery, p. 81–86 (2017).

[12] Einarsson, A. F., Patreksson, P., Hamdaqa, M. and Hamou-Lhadj, A.: SmartHomeML: Towards a Domain-Specific Modeling Language for Creating Smart Home Applications, *in Proc. of IEEE International Congress on Internet of Things (ICIOT)*, pp. 82–88 (2017).

[13] Einarsson, A. F., Patreksson, P., Hamdaqa, M. and Hamou-Lhadj, A.: SmarthomeML: Towards a domain-specific modeling language for creating smart home applications, *in Proc. of IEEE International Congress on Internet of Things (ICIOT)*, IEEE, pp. 82–88 (2017).

[14] Samuel, S. S. I.: A review of connectivity challenges in IoT-smart home, *in Proc. of MEC International conference on big data and smart city (ICBDSC)*, pp. 1–4 (2016).

[15] Cicirelli, F., Fortino, G., Giordano, A., Guerrieri, A., Spezzano, G. and Vinci, A.: On the design of smart homes: A framework for activity recognition in home environment, *Journal of medical systems*, Vol. 40, No. 9, p. 200 (2016).

[16] Guan, J. and Offutt, J.: A model-based testing technique for component-based real-time embedded systems, *in Proc. of IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 1–10 (2015).

[17] Peter, S. and Givargis, T.: Component-Based Synthesis of Embedded Systems Using Satisfiability Modulo Theories, *in Proc. of ACM Trans. Des. Autom. Electron. Syst.*, Vol. 20, No. 4 (2015).

[18] Sultan, M. and Ahmed, K. N.: SLASH: Self-learning and adaptive smart home framework by integrating IoT with big data analytics, *in Proc. of Computing Conference*, pp. 530–538 (2017).

[19] Nandi, C., Monot, A. and Oriol, M.: Stochastic contracts for runtime checking of component-based real-time systems, *in Proc. of International ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE)*, pp. 111–116 (2015).

[20] Yamamoto, T., Hara, T., Ishikawa, T., Oyama, H., Takada, H. and Azumi, T.: Component-Based mruby Platform for IoT Devices, *Journal of Information Processing*, Vol. 26, pp. 549–561 (2018).

[21] Doan, T. T., Safavi-Naini, R., Li, S., Avizheh, S. and Fong, P. W.: Towards a resilient smart home, *in Proc. of the 2018 Workshop on IoT Security and Privacy*, pp. 15–21 (2018).

[22] Banerjee, A., Sufyanf, F., Nayel, M. S. and Sagar, S.: Centralized framework for controlling heterogeneous appliances in a smart home environment, *in Proc. of International Conference on Information and Computer Technologies (ICICT)*, IEEE, pp. 78–82 (2018).