ECTI Transactions on Computer and Information Technology
Journal homepage: https://ph01.tci-thaijo.org/index.php/ecticit/
Published by the ECTI Association, Thailand, ISSN: 2286-9131

ECHONET Lite Framework Based on Embedded Component Systems

JieYing Jiang[1], Feng Qi[2], Hiroshi Oyama[3], Hiroaki Nagashima [4], and Takuya Azumi[5]

ABSTRACT

The large number and many types of electrical equipment in modern households pose major challenges to developers. To increase the breadth of smart homes, improving the development efficiency of electrical equipment in smart homes is essential. The present research proposes a development method based on embedded components for devices in smart homes. Besides improving the development efficiency, this method of development reduces the developmental complexity and increases the scalability of electrical equipment. The smart home devices are developed and controlled in TOPPERS Embedded-Component System (TECS), a component description language that automatically generates template C files when expanding new components or functions. Using the ECHONET Lite plugin, TECS then reads the parameters of the electrical device in the device description file (JSON file), and outputs the function parameters for implementation by the developer. The evaluation shows that the code automatically generated by the plugin slows down the software execution time by only 2% compared with the default, thus improving the software development efficiency.

Article information:

Keywords: Non-orthogonal Multiple Access (NOMA), Backscatter, Outage Probability

Article history:
Received: April 23, 2021
Revised: October 11, 2021
Accepted: December 6, 2021
Published: March 12, 2022 (Online)

DOI: 10.37936/ecti-cit.2022161.245976

1. INTRODUCTION

Recently, due to the rapid development of smart homes, the scale of embedded control system software has been continuously expanding. The number and types of electrical equipment are a heavy development burden for developers [11]. Rapid device updates are relatively heavy burden for developers. 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. It is significant for developers to find effective development methods to improve the efficiency and reusability of smart home software development.

Therefore, we propose a development method that developers will find convenient to maintain and expand embedded systems. Reducing design complexity is particularly important for rapid development and ensuring correct software products. 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. Using high-level languages for development can effectively improve the portability and reusability of application software programs. Component-based development (CBD) is particularly suitable for this purpose [16]. CBD is a process that emphasizes the use of reusable software components for design and development. Developers can quickly build application software by adding or removing components as needed, to provide greater flexibility and simplify development and maintenance.

To make user's lives more convenient, with the support of smart homes, the electrical equipment in the home is developed to facilitate its easy usage. The main goal of IoT is to advance a better and safe society, where "Everything is a service," such as public safety, environment, health care and production [2].

[1,2,5]The authors are with Graduate School of Science and Engineering, Saitama University, Japan, E-mail:jan.k.413@ms.saitama-u.ac.jp, qi.f.889@ms.saitama-u.ac.jp and takuya@mail.saitama-u.ac.jp
[3]The author is with OKUMA Corporation, Japan, E-mail: hiroshi.oyama@mhm-global.com
[4]The author is with Cores Co., Ltd., Japan, E-mail: nagasima@core-s.co.jp

With the implementation of integration technology, communication technology, interoperability and cabling standards, smart home network will continue to improve. This includes the operation and management of all smart furniture, equipment, and systems in home networks using integrated technologies. The technical characteristics of the smart home are as follows. Through the home gateway and its system software, a smart home platform system is established. An intelligent home terminal uses computer technology, microelectronic technology, and communication technology to integrate all the functions of the home intelligence. Thus, a smart home is built on an integrated platform. The external expansion components realize the connect to and control of the household appliances. The development board and all the components are connected through an embedded system.

In this research, we propose a component-based smart home development framework based on TOPPERS Embedded-Component System (TECS). The framework first defines the attributes and functions of the device by designing application modules in the component description language. The TECS generator then creates the necessary header files and template C files. The device parameters in the ECHONET Lite device description file (JSON file) are obtained through the ECHONET Lite plugin. To write code to support the device, developer only need to add the function content in the C file based on the device parameters. Finally, the device application module is compiled. The proposed framework generates a binary file for the application modules, and writes it into the embedded board. The smart home is connected and controlled through the general-purpose input/output interface of the embedded board.

In our proposed framework, a platform can be developed outside of the smart home development environment. The proposed framework improves the scalability of smart home systems and reduces the difficulty of development and maintenance. As the framework is highly independent, device functions are easily added or removed. The remainder of this paper is organized as follows. Section 2 introduces the system model, including TECS and ECHONET Lite. Section 3 describes the process and method of realizing a smart home in the proposed framework. Section 4 discusses related frameworks and components for smart home development. Section 5 evaluates the proposed framework, and Section 6 suggests future work and research directions.

2. SYSTEM MODEL

This section describes the system model of the proposed framework (Fig. 1). Section 2.1 introduces the basic information and the basic development framework based on TECS. Section 2.2 introduces ECHONET Lite.

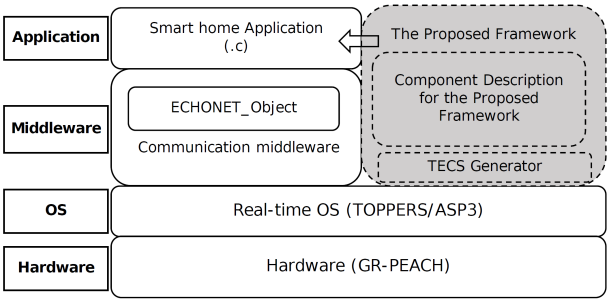

Fig.1: *System Model Of The Proposed Framework.*

2.1 TOPPERS Embedded Component System

TECS is a component system designed for developing embedded systems [11]. The TECS-based development method is to first divide the project into components and subsystems, and then develop application components by using subsystems. This method of development can improve the reusability of the embedded software, improving production efficiency and reducing development costs. TECS can be used in domains of embedded systems because it supports multi-sized and diverse components [14].

TECS has statically developed components, which will not generate any excessive overhead when running, effectively reducing the memory resource requirements. The control of a smart home and the detection of its status is performed in real-time, and thus we also develop components based on a real-time system. TECS can deal with real-time operating system (RTOS) resources (such as tasks and semaphores) as components [13]. The TECS generator plugin can componentize RTOS features, requiring complex static API generation using the proposed plugin [15]. Thus, TECS meets the requirements of smart home development. The TECS design is as follows.

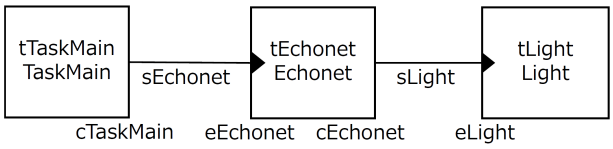

Fig.2: *Component Diagram.*

2.2 Component model

An example of a component is shown in Fig. 2. 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.

2.3 Component description

We use the component description language (CDL) to describe the components in TECS. In TECS, a component contains a *signature*, *celltype*, and single *cell* description specified using CDL.

2.3.1 Signature description

The function of the signature description 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 (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]."

```
1 signature sEchonet{
2   ER onOffPropertySet ([in] const uint8_t * src, [in] int
        size, [out] bool_t * anno);
3 };
4 signature sLight{
5   void setOperatingStatus_ON();
6   void setOperatingStatus_OFF();
7 };
```

Fig.3: *Signature Description.*

2.3.2 Celltype description

A Celltype description defines a component's *celltype*, including the component's *entry port*, *call port*, attributes, and variables which is 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., cEchonet). The definition of the entry port is similar to that of the *call port*. The attributes and variables are defined using keywords "attr" and "var," followed by a name.

```
1  celltype tEchonet{
2   call sLight cEchonet;
3   entry sEchonet eEchonet;
4    attr {
5    uint8_t versionInformation;
6    uint8_t identificationNumber;
7    uint8_t manufactureCode;
8    uint8_t setPlace;
9   };
10 };
11 celltype tLight{
12  entry sLight eLight;
13  attr {
14   uint8_t  propertyCode;
15   ATR   propertyAttribute;
16   uint8_t  propertySize;
17   intptr_t extendedInformation;
18  };
19 };
```

Fig.4: *Celltype Description.*

2.3.3 Cell description

A *Cell* is used to instantiate and connect which is shown in Fig. 5. The keyword "*cell* " is followed by the *celltype* e.g., Echonet, and name. The *cells* are linked by specifying the *call port*, caller name, and *entry port*, in the same order. For example, in Fig. 5, the eLight *entry port* of the Light *cell* is connected to the cEchonet *call port* of the Echonet *cell*.

```
1 cell tEchonet Echonet{
2  cEchonet = Light.eLight;
3  versionInformation = 0x82;
4  identificationNumber = 0x83;
5  manufactureCode = 0x8A;
6  setPlace = 0x00;
7 };
```

Fig.5: *Cell Description.*

2.3.4 Development flow

The development process when using TECS is shown in Fig. 6. Note that the TECS development can be divided into two modules: 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. The application diagrams and predefined *celltype* are used to develop applications. The application module is generated by connecting the header file, and compiling the interface, and *celltype* code.

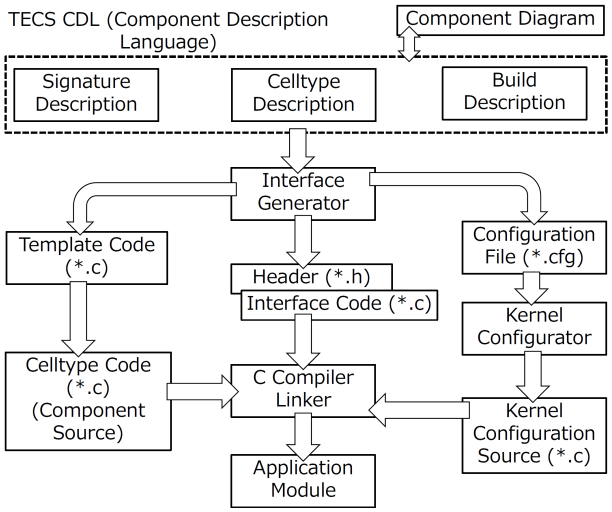

Fig. 6: *Development Flow.*

2.4 ECHONET Lite

ECHONET [9] has been certified as the home network standard by the International Electrotechnical Commission and the International Organization for Standardization. ECHONET Lite standards are structured as shown in Fig. 7. Besides providing simple remote control commands (such as the "ON/OFF" button), ECHONET device objects standardize the functions of highly complex devices, enabling the required advanced control of energy management applications [5]. However, widespread use of the ECHONET protocol has been prevented by two limitations. First, ECHONET specification requires a relatively complex system configuration with multiple controllers and multiple devices. Second, development must comply with the overall complexity of ECHONET protocol. To mitigate these limitations, the ECHONET protocol has been substantially simplified, and was reissued as the ECHONET Lite protocol in 2011.

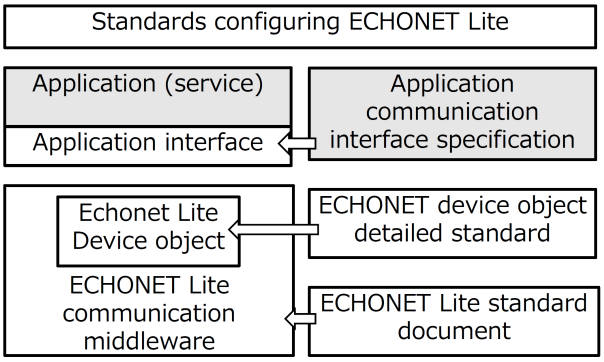

Fig. 7: *Structure Of The ECHONET Lite Standards.*

The ECHONET Lite protocol provides a standard method for controlling household appliances. In particular, the ECHONET Lite protocol achieves interoperability between devices from different manufacturers in the smart home. Although ECHONET Lite is based on ECHONET standards, its structure enables easy development by home network system builders and service system developers. Devices that comply with the ECHONET Lite specification cannot be connected to devices complying with the ECHONET specifications, but may coexist in the same system.

The system architecture of ECHONET Lite is shown in Fig. 8. The largest manageable area of ECHONET Lite is the domain, which specifies the variety of controlled resources (e.g., home equipment, appliances and consumer electronics, sensors, controllers, and remote controls) within the specified network range of ECHONET Lite. A system performs communication and linked operations between the devices and controllers that monitor/control/operate the devices themselves. Multiple systems that reside in a domain can include the same device or controller. When connecting a system to another system outside the domain, the ECHONET Lite gateway can be used as an interface.

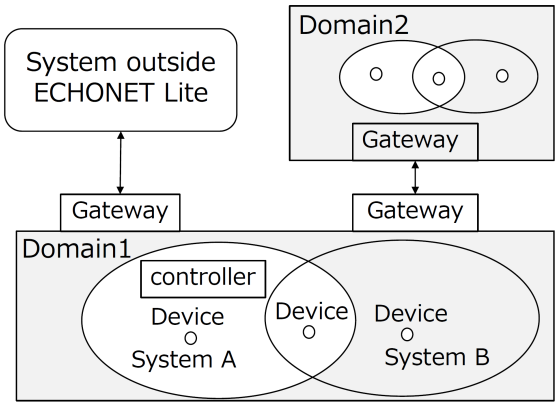

Fig. 8: *Architecture Of The ECHONET Lite System.*

The ECHONET Lite standard defines the communication between components (called ECHONET objects). An ECHONET object has certain properties that dictate the operation of actual devices. This middleware provides 1) a static application interface (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 code in the TOPPERS kernel configurator. The code is then compiled and linked to the application software.

2.5 Embedded board

This subsection describes the design and method of smart home development based on TECS. Functions are implemented on a GR-PEACH development

board with the specifications listed in Table 1. To evaluate the sending and receiving of data, we connect the board to a host PC via a LAN cable and evaluate the data sent and received.

Table 1: *Specifications Of The Board.*

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

3. DESIGN AND IMPLEMENTATION

This section introduces smart home development methods and presents our developmental framework.

This section is organized as follows. Section 3.1 introduces the development method of the smart home. Section 3.2 describes the process and method of the proposed framework. Section 3.3 introduces the device description in JSON file and Section 3.4 introduces ECHONET Lite plugin.

3.1 The development method of smart home

The hardware implementation of a multi-platform control system for home automation was proposed by Kadima [7]. Such systems belong to a wider domain commonly called smart home systems. The Internet of Things (IoT) home automation systems equipment can be designed on a Raspberry Pi microcomputer, and light bulbs and circuits can be connected on an Arduino Uno board as the controller and relay. The signals are transmitted through radio frequency waves (in wireless communication) or through wires (in wired communication). In this setup, IoT is easy to use and the complexity is reduced since a single device controls various other devices. Additionally, hardware and software technologies are combined and can be easily used in the automation applications of smart homes.

The SmartHomeML based modular language is a domain-specific modeling language for smart home applications [1]. SmartHomeML uses model-to-text conversion templates to generate smart home adaptation services that meet the target provider's specifications. Users of SmartHomeML can define new skills, and can apply template-based model-to-text conversion for automatic adapter generation and connection to smart home devices. SmartHomeML can also be used on other development platforms, such as the Amazon Alexa platform. SmartHomeML provides the mapping relationship between the module language and development platform, reducing the difficulty of development, although sorting the mapping relationship is a time-consuming task.

The proposed development framework is based on embedded-component development, and is extendible to local smart home devices. Our development method has a more transparent process and structure than the methods of Kadima [7] and Einarsson [1]. When extending the functions of any equipment belonging to the Energy Conservation and Homecare Network ECHONET Lite protocol, we can familiarize ourselves with the mapping relationship between the module language and development platform. The generator can create the *celltype* code using the template code, and then use the template code to implement the functions.

3.2 Proposed framework

The proposed framework is shown in Fig. 9.It shows how software for smart homes is developed based on device descriptions in JSON. The JSON code contains the version information, status information, and function information of the smart home. As the present example is limited to lighting, we extracted the light-related parameters from the ECHONET Lite device description JSON file.

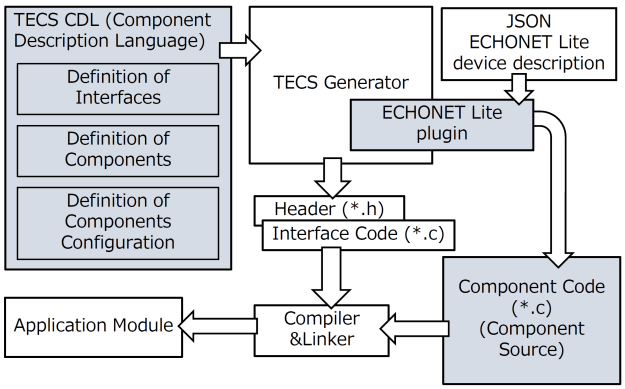

Fig. 9: *Architecture Of The ECHONET Lite System.*

Once the components have been defined, the TECS generator generates a header file and interface code. The generated C file contains template code for calling all header functions in the generated file.

The ECHONET Lite plugin (currently under development) acquires the information about ECHONET Lite and light components from the TECS generator, then acquires the related parameters of the smart home from the JSON file, and finally creates the TECS component code. After compiling the generated interface code and TECS code, a binary file that can be written to the embedded board is generated.

After writing the binary file to it, the embedded board is connected to the computer via USB and LAN. Finally, the IP address of the embedded board is entered into the super speed node generator (SSNG) for ECHONET Lite application. The user controls the write operations through the SSNG.

3.3 Device Description in JSON file

In the ECHONET Lite device description, the device specification part (in JSON format) provides the device descriptions of objects such as air conditioners and smart meters. The JSON file defines the relevant parameters of the devices supported in ECHONET Lite. Before developing the device, the developer can learn the functions and device parameters of the developed device in detail through the JSON file.

As shown in Fig. 10, this code is a part of the device description JSON file, and specifies some attributes of the device which are shown in this code. For example, 0x80 represents the action's state of the device. The action state includes two states: one on, the other off. The *on* and *off* states are represented by 30 and 31, respectively.

```
1  "devices":{
2   "0x0EF0":{
3    "validRelease":{"from":"A","to":"latest"},
4    "className":{"en":"Node profile"},
5    "elProperties":{
6     "0x80":{
7      "validRelease":{"from":"A", "to":"latest"},
8      "propertyName":{"en":"Operating status"},
9      "accessRule":{"get":"required", "set":"optional", "
           inf":"required"},
10     "data":{"$ref":"#/definitions/state_ON-OFF-3031"}
11       }
12     }
13   }
14 }
```

Fig. 10: *ECHONET Lite Device Description.*

The ECHONET Lite plugin extraction of device parameters from the device description file is shown in Fig. 12. The main function of this code is extracting and printing the device parameters in the selected device description file. The equipment to be printed and its functional parameters can be selected by the user. The printed parameters are shown in Fig. 13.

The function name generated from the JSON file is shown in Fig. 14, including the parameter part of the function.

The component code automatically generated based on the JSON file is shown in Fig. 15. Each component needs to use different GPIO interfaces. The specific functional part of the component code needs to be written manually.

The relevant function parameters of the developed equipment can be applied along with the interface code in the code development. After the final compilation, the required application module files will be generated.

```
1 devdesc_json_fname = "appendix_v3-1-6r5/
      EL_DeviceDescription_3_1_6r5.json"
2 devdesc_json = File.read(devdesc_json_fname)
3 DevDesc = JSON[devdesc_json]
4 Devices = DevDesc["devices"]
5 Definitions = DevDesc["definitions"]
```

Fig. 11: *Device Description File Selection.*

3.4 ECHONET Lite plugin

The ECHONET Lite plugin extracts the relevant parameters of the device from the ECHONET Lite device description file. In the initial development period, developers require time to understand the device and implementations of the various description types and the many parameters of the related device functions. Developers can use the parameters extracted by the ECHONET Lite plugin in their models, and draw component model diagrams of TECS.

The procedure for selecting the device description file by ECHONET Lite plugin is shown in Fig. 11. The filename is the name of a JSON file, and is represented by JSON fname. The device description file is designated as DevDes. "Devices" and "Definitions" represent the classes of the device and the device definition, respectively, in the device description. Devices represent the class of the device in the device description file. To facilitate the development and debugging of devices, this code provides different device description demonstration files.

```
1  def print_properties( val, indent="   " )
2   val.each{|prop_id, val2|
3    if val2['oneOf'] then
4     print( "#{indent}oneOf\n" )
5      val2['oneOf'].each{ |val3|
6       print( "#{indent}    #{val3['propertyName']['en
            ']} = #{prop_id}\n" )
7       }
8     else
9      print( "#{indent}#{val2['propertyName']['en']} =
          #{prop_id}\n" )
10     if val2['data'] && val2['data']['$ref'] then
11       ref = val2['data']['$ref'].sub( /#\/definitions\//,
           "" )
12       val3 = Definitions[ref]
13     if val3['type'] == "state" then
14       print "#{indent}  state_type = '#{ref}'\n"
15       print_state_type( indent + "     ", val3 )
16     elsif val3['type'] == "number" then
17       print "#{indent}  number_type = '#{ref}'\n"
18       print_number_type( indent + "     ", val3 )
19       end
20     else
21      print "#{indent}  data or ref not defined\n"
22     end
23   end
24   }
25 end
```

Fig. 12: *ECHONET Lite Plugin.*

```
Node profile = 0x0EF0
Operating status = 0x80
 state_type = 'state_ON-OFF-3031'
 ON = 0x30
 OFF = 0x31
Version information = 0x82
Identification number = 0x83
Fault status = 0x88
Fault description = 0x89
Manufacture code = 0x8A
Business facility code = 0x8B
Product code = 0x8C
Production number = 0x8D
Production date = 0x8E
```

Fig.13: *Printed Description Information Of The Device.*

```
int getpropertymapPropSet ([in] const EPRPINIB *item, [in] const void *src, [in] int size, [out] bool_t *anno)
int versioninformationPropSet ([in] const EPRPINIB *item, [in] const void *src, [in] int size, [out] bool_t *anno)
```

Fig.14: *Automatically Generated Function Name.*

```
void OnTimerSetting_prop_set ([in] const EPRPINIB *item, [in] const void *src, [in] int size, [out] bool_t *anno)
{
 if(size! = 1)
  return 0;
 *anno = *((unint8_t*)item->exinf) != *((unint8_t*)src);
 switch (*(unint8_t*)src) {
  case 0x30: cGeneralLighting_SetON( );
   break;
  case 0x31: cGeneralLighting_SetOFF( );
   break;
 default:
  return 0;
 }return 1;
}
```

Fig.15: *Component Code.*

4. RELATED WORK

This section introduces the development and design of mainstream smart home system models. Development platforms and methods for smart homes are varied and include integrated modeling languages [1], independent custom development [12], and design and implementation on open-source platforms [4]. Table 2 compares the features of our work with those of current component development methods for smart home software.

Table 2: *Ours Versus Previous.*

	EBD	ML	EX	CBD	MA
Component + TECS [15]	√	√		√	
Test Model [6]	√	√		√	
TECSInfo [11]	√	√		√	
Customized Design [7]			√		
CoDeL [10]	√		√	√	
SmarthomeML [1]		√			
Design by Contract [3]				√	
ASP+TECS [13]	√	√		√	
MIMO + LQR [8]				√	
mruby on TECS [16]	√	√		√	√
This Thesis	√	√	√	√	√

EBD: Embedded-based development
ML: Modeling language
EX: Extendability
CBD: Component-based development
MA: Maintainability

The holistic framework of Biljana et al. incorporates different components from the IoT architectures/frameworks proposed in the literature. This framework efficiently integrates smart home objects in a cloud-centric IoT based solution to improve the integrability of the existing state-of-the-art applications for smart homes into IoT enabled environment. The authors identified a smart home management model for the proposed framework and the main tasks to be performed at each level. In addition, they discussed the practical design challenges with an emphasis on data processing, as well as smart home communication protocols and their interoperability. They contend that their holistic framework provides a solid base for future developers of IoT based smart home solutions.

Shirata et al. proposed a component framework for obtaining the information about component systems [11]. The proposed framework supports obtaining static information of the generated components and runtime component information during generation and execution periods. The framework uses a plugin to automatically perform this operation, automatically generating a unit that saves the information of the system. Plugin and dynamic connections can effectively reduce the number of lines of code, 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 [16], 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 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.

Guan et al. presented a novel integration test strategy (CREMTEG) for component-based real-time embedded software [6]. 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 finding the difference between the expected and actual diagnosis results easily.

Einarsson et al. introduced SmartHomeML [1], a domain-specific modeling language for smart home applications, which allow users to define new functionalities. SmartHomeML has 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.

Mukendi et al. proposed the design and implementation of a customized IoT smart home [7]. 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 a 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 each 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 LCD screen.

Nandi et al. presented a statistical inference based approach for computing real-time contracts for component-based real-time control applications [3]. 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) [10], 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 [8]. The framework utilizes a multiple-input multiple-output controller that coordinates the reserve size to track the dynamic resource requirements of the software components.

5. EVALUATION

This section evaluated the application execution time. We evaluated the application execution time for 1000 times and calculated their average execution time. First, the execution time of the handwritten code version of the application, which is about 1.20 microseconds, was evaluated. Then, the application execution time of the automatically generated code version, which is about 1.22 microseconds, was evaluated. The execution time of the automatically generated code is about 0.02 microseconds longer than the execution time of the handwritten code. The automatically generated code is only 2% slower than the execution time of the handwritten code. Therefore, the results show that the automatically generated code can be used in practical applications. The comparison of results is shown in Fig. 16.

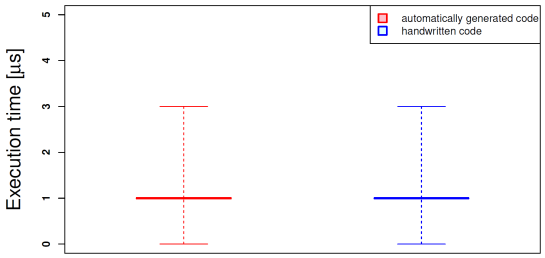

Fig. 16: *Application Execution Time.*

6. CONCLUSIONS

This study proposed a development method based on embedded components (i.e., CBD) to develop devices for smart homes. Through the embedded development technology, the software and hardware development for 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. The established development model can help developers clarify the development structure. The parameters in the device description file can be obtained through the ECHONET Lite plugin, and the developer will gradually improve the code according to the interface code and parameters. The software for smart homes can be gradually developed using the established model diagram, reducing the complexity of development.

The proposed developmental framework enables the development of embedded components for smart homes. The framework can reduce the complexity of development and improve the scalability of smart homes, and promote the update and maintenance of device functions. The proposed framework enables automatically generated based on the device description file. The framework is based on ECHONET Lite for device development, but currently cannot support

other protocols.

In future work, we will extract the relevant parameters from the device description file through the ECHONET Lite plugin, and combine them with the interface code for cross-compilation. An automatically generated application module is also on the agenda.

References

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

[2] B. L. Risteska. Stojkoska and K. V. Trivodaliev, "A review of internet of things for smart home: Challenges and solutions," *Journal of Cleaner Production*, vol. 140, pp. 1454–1464, 2017.

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

[4] F. Cicirelli, G. Fortino, A. Giordano, A. Guerrieri, G. Spezzano and A. Vinci, "On the design of smart homes: A framework for activity recognition in home environment," *Journal of medical systems*, vol.40, no.9, pp.200, 2016.

[5] H. Kodama, "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.

[6] J. Guan and J. Offutt, "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*, pp. 1–10, 2015.

[7] M Nathan. Kadima and F. Jafari, "A customized design of smart home using internet-of-things," in *Proc. of the 9th International Conference on Information Management and Engineering*, pp. 81–86, 2017.

[8] N. Khalilzad, M. Ashjaei, L. Almeida, M. Behnam and T. Nolte, "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*, pp. 1–10, 2015.

[9] P. Cuand, Y. Makino, L. Yuto and T. Yasuo, "Semantic service gateway for ECHONET based smart homes," in *Proc. of Conference on Innovation in Clouds, Internet and Networks and Workshops*, pp. 175–179, 2019.

[10] S. Peter and T. Givargis, "Component-based synthesis of embedded systems using satisfiability modulo theories," *ACM Transactions on Design Automation of Electronic Systems*, vol.20, no.4, pp. 1–27, 2015.

[11] S. Shirata, H. Oyama and T. Azumi, "Run-time component information on embedded component systems," in *Proc. of International Conference on Embedded and Ubiquitous Computing*, pp. 166–173, 2018.

[12] S. S. I. Samuel, "A review of connectivity challenges in IoT-smart home," *2016 3rd MEC International Conference on Big Data and Smart City (ICBDSC)*, pp. 1-4, 2016.

[13] T. Azumi, H. Takada, T. Ukai and H. Oyama, "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*, pp. 151–155. IEEE, 2010.

[14] T. Azumi, M. Yamamoto, Y. Kominami, N. Takagi, H. Oyama and H. Takada, "A new specification of software components for embedded systems," in *Proc. of IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, pp. 46–50, IEEE, 2007.

[15] T. Kawada, T. Azumi, H. Oyama and H. Takada, "Componentizing an operating system feature using a TECS plugin," in *Proc. of the 4th IEEE International Conference on Cyber-Physical Systems, Networks, and Applications*, pp. 95–99. IEEE, 2016.

[16] T. Yamamoto, T. Hara, T. Ishikawa, H. Oyama, H. Takada and T. Azumi, "Component-based mruby platform for IoT devices," *Journal of Information Processing*, vol.26, pp.549–561, 2018.

Jieying Jiang He received his B.E. degree from Southwest University of Science and Technology in 2014. His research interests include real-time and embedded systems.

Feng Qi He received his B.E. degree from Hefei University of Technology in 2016. His research interests include embedded component systems.

Hiroshi Oyama He received his Dr. Engineering degree from Gifu University in 2002. He works at Okuma Corporation as a Senior Engineer and also works at Nagoya University as a Visiting Professor. He is studying software components and programming language for embedded systems.

Hiroaki Nagashima He works at Cores Co., Ltd. as a embedded software developer.

Takuya Azumi He is an Associate Professor at the Graduate School of Science and Engineering, Saitama University. He received his Ph.D. degree from the Graduate School of Information Science, Nagoya University. From 2008 to 2010, he was under the research fellowship for young scientists for Japan Society for the Promotion of Science. From 2010 to 2014, he was an Assistant Professor at the College of Information Science and Engineering, Ritsumeikan University. From 2014 to 2018, he was an Assistant Professor at the Graduate School of Engineering Science, Osaka University. His research interests include real-time operating systems and component-based development. He is a member of IEEE, ACM, IPSJ, and IEICE.