Runtime knowledge graph based approach to smart home application development

Minchen Zhu[1,2], Xinshu Ye[1,2], Tao Xiang[1,2], Yun Ma[3], Xing Chen[1,2]*
[1]College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350116, China
[2]Fujian Key Laboratory of Network Computing and Intelligent Information Processing, Fuzhou 350116, China
[3]School of Software, Tsinghua University, Beijing 100084, China
*Corresponding author: chenxing@fzu.edu.cn

***Abstract*—Smart home is an important application area of the Internet of things (IoT). However, the diversification of smart home application scenarios increases the difficulty of understanding the scenarios for developers. And the heterogeneity of the programming interfaces of smart devices as well as the close coupling of the code to the underlying systems is still an important work for the developers. Furthermore, the complexity and variability of business requirements poses a great challenge to the development of applications logic. In this paper, we present a runtime knowledge graph based approach to smart home application development. First, a conceptual model describing the smart home scenarios is defined. Second, the manageability of smart devices is abstracted as runtime knowledge graphs that are automatically connected with the corresponding systems. Last, a method of automatically generating smart home applications is proposed. Our approach can reduce code by about 85 percent at least, and an experiment on a real-world application scenario demonstrates the feasibility, effectiveness, and benefits of the new approach to smart home application development.**

Keywords—Internet of things; models at runtime; software architecture

I. INTRODUCTION

Trillions of Internet of things (IoT) devices, such as smart wearable devices, smart robots and smart appliances, will connect to the network and create massive amounts of data. Smart home has become an important field for the IoT. However, smart home application development is difficult and complex, which brings great obstacles to the application and promotion of smart home.

From the view of system implementation, applications access various information through the programming interface of the smart home device and analyze, process, and feedback this information for specific smart services and residential conditions. In current smart home application development, programming devices are typically carried out very close to the operating system, therefore requiring developers to focus on low-level system issues. This not only distracts developers from application logic, but also requires a technical background rarely found among application domain experts. Therefore, developers face many challenges.

First of all, smart home applications are developed for a variety of scenarios. The diversity of devices makes smart home services more and more diversified, while making smart home scenarios more complex and diverse. The diversity of scenarios puts forward a challenge for developers to understand the scenarios.

Secondly, smart home applications are developed for heterogeneous devices. The type of devices data is inconsistent, and communication between devices is difficult. Handling the heterogeneity of the programming interfaces as well as the close coupling of the code to the underlying system is still an important challenge for the developers.

Thirdly, smart home applications are developed for a variety of needs. The complexity and variability of business requirements poses a great challenge to the development of applications logic.

Because of the above challenges for most developers, it is difficult to overcome. Therefore, it is slow in the smart home scenario application development. It is necessary to provide an approach that that can meet the above challenges.

MDE [1] is a software engineering methodology that focuses on creating and exploiting domain models (i.e., abstract representations of the knowledge and activities in a specific domain). MDE aims to address the complexity of software development, by raising the abstraction level and automating the generation of the application artifacts. Runtime models have been widely used in different systems to support data manipulation, self-repair [2], and dynamic adaption [3–5].

In this paper, we present a runtime knowledge graph based approach to smart home application development. First, a conceptual model describing the smart home scenarios is defined. Second, the manageability of smart devices is abstracted as runtime knowledge graph that are connected with the corresponding systems automatically. Last, a method of automatically generating smart home applications is proposed.

The whole approach only needs to provide a description of the current smart home scenario, configure the requirements files, define a group of meta-models and mapping rules, thus greatly reducing the workload of hand coding. All of the automatically generated application logic can be carried out through executing programs on the runtime knowledge graph. When the demand changes, the developer only needs to manage the configuration files to realize the rapid development of the scenario. Developers need not pay attention to the devices and data, only need to focus on the demands.

978-1-5386-7244-0/18/$31.00 ©2018 IEEE
DOI 10.1109/ICIOT.2018.00022

We organize the rest of this paper as follows. Section 2 introduces related work. Section 3 overviews our approach with a motivating example. Section 4 presents a runtime knowledge graph based approach. Section 5 gives the implementation of the approach and reports the evaluation. Section 6 concludes this paper and indicates our future work.

II. RELATED WORKS

The modeling approach is widely used by many researchers in the smart home application development. The choice of a modeling language in software engineering is traditionally restricted to the tools and meta-models invented specifically for that purpose. On the other hand, semantic web standards are intended mainly for modeling data, to be consumed or produced by software. However, both spaces share enough commonality. Shoko Nakamura [6] developed and proposed a smart home system to improve the quality of human life. They apply a simple model to represent the activities of home residents, the operations of appliances, and energy sources. They also designed a model to represent such strategies and criteria quantitatively. Daniel Lüddecke [7] presented an approach of modeling contextual information of a context-aware system using the example of a context-aware in-car infotainment system, and described the requirements of the scenario in detail and verifies the necessity of Web Ontology Language (OWL) as a modeling language. Dominique Ernadote [8] presented an extension of the MMP approach [9] consisting in the mappings of dedicated ontologies to standard metamodels. The approach presents the advantage of exposing modeling data in different perspectives appropriate to the involved stakeholders. Borislav Iordanov [10] described their experience using OWL as the language for Model-Driven Development (MDD). The work of the researchers above validates the feasibility of using knowledge graphs to model smart home scenarios. However, their work does not focus on the relationship between the devices and the user's needs, nor does it provide an approach to develop the application.

In our previous work [11], we proposed a runtime model based approach to IoT application development. Developers only need to construct runtime models of sensor devices and a customized model to satisfy personalized application scenarios. Operations on the customized model are automatically mapped to the ones on sensor device runtime models through model transformation and all the application logic can be carried out by executing operating programs on the customized model.

The following paragraph focuses on our prior work in the field of model driven engineering. For a given meta-model and a given set of management interfaces, SM@RT [12] can automatically generate the code for mapping models to interfaces with su□cient runtime performance. If users change the meta-model, SM@RT can re-generate the mapping code. More details can be found in our previous work [13]. In addition, for the situation of incomplete formalized modeling languages, our previous work [14] provides an meta object facility (MOF) meta-model extension mechanism with support for upward compatibility and automatically generates a model transformation for model integration, and our work on architecture-level fault tolerance [15] can also compensate for this to a degree. However, it is difficult to construct runtime models to satisfy the smart home application development requirements from scratch. The approach in this paper is built on our previous work.

III. APPROACH OVERVIEW

In this section, we give an overview about our approach. We first present a motivating example, which illustrates the challenges encountered in the development of smart home scenarios. Then we briefly introduce a runtime knowledge graph based approach to smart home application development.

A. *Motivating example*

There are three rooms in the house in Fig. 1, each room has some smart devices. Among them, there are some smart planting device-related devices, such as smart temperature control devices, smart water valves, smart lights and smart curtains. The suitable temperature, humidity and light intensity required for plant growth are provided by these smart devices.

According to the previous smart home scenario development approaches, developers develop IoT systems based on smart devices for different regions and different devices. However, this can lead to some common programming difficulties.

Programming difficulties are reflected in the following aspects. On the one hand, application scenarios may consist of different types of device that need to be collaboratively managed. Developers must be familiar with the APIs provided by smart devices and build programs upon them. On the other hand, developers need to develop bottom-up for each scene. Developers must write many different programs to manage similar scenarios. In addition to programming difficulties, changes in requirements and changes in devices may cause applications to be redeveloped.

Taking the application development of smart planting scenario as an example, developers face the following three challenges in developing applications:

Firstly, the demands for smart planting scenario is complex. Developers need to take into account the different growth needs of the chlorophytum planted in area A and the monster planted in area B. In addition, different devices, light intensity, soil moisture, etc., need to be considered. It's

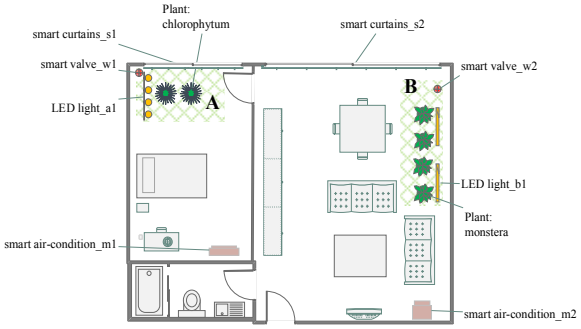

Figure 1. The motivating example

not always easy for developers to pinpoint all the objects involved in the smart planting scenario application.

Secondly, there are many complicated smart devices in the smart planting scenario. Developers need to be familiar with the programming interfaces for smart devices such as the interfaces of smart water valves, smart lights, and smart curtains. Application development is done at a low-level, very close to the operating system, requiring developers to focus on low-level system issues. The underlying API can be very complex and the amount of data collected may be large. As a developer, this can be hard to deal with.

Finally, the business logic of applications for smart plantings is complex and changeable. The logic of the application also changes when the location of the device changes or when the type of plant changes. Developers need to re-develop the application scenario. This uncertainty creates a lot of unnecessary and repetitive consumption of application development.

B. Our approach

Our approach makes smart home scenario application development easier and more efficient. We used a model-driven approach to development. In this regard MOF meta-modeling architecture [16] defining four modeling layers, from M3 meta-meta model layer to M0 instance layer, proved useful. Modeling languages are typically specified at M2 (meta-model layer). Once they are used to describe models reflecting reality, M1 model layer is populated. When the models at M1 level are instantiated M0 level is reached.

Fig. 2 shows an overview of runtime knowledge graph based smart home application development. Firstly, conceptualize the device layer and abstract the objects in smart home scenarios from the top. Secondly, construct the runtime knowledge graph of the smart devices to provide unified management for application development. Thirdly, the logic modeling program is automatically generated according to the demands of the users, and is applied to the runtime knowledge graph layer, and finally development of the smart home scenario application is realized.

In our approach, developers develop smart home scenario applications by configuring requirement files, thus dramatically reducing the manual coding effort.

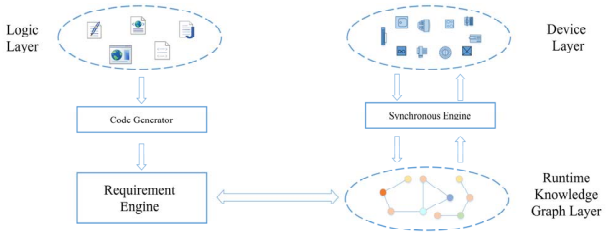

Figure 2. Overview of the runtime knowledge graph based smart home application development

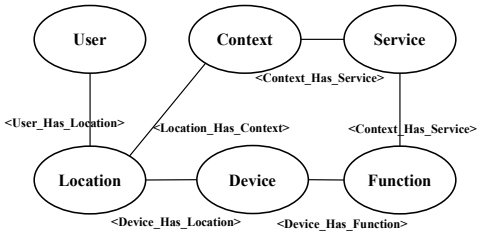

Figure 3. The conceptual model of smart home application scenarios

IV. OUR APPROACH TO SMART HOME APPLICATION DEVELOPMENT

This section introduces the runtime knowledge graph based approach to smart home application development. The definition of the smart home conceptual model is the basis for constructing the runtime knowledge graph of the smart home scenario. The construction of runtime knowledge graph is conducive to the automatic generation of the code of the smart home scenario application.

A. The definition of conceptual model of smart home scenario

We built the conceptual model of smart home application scenarios for developers. Fig. 3 shows the conceptual model of the smart home scenario. The device layer in Fig. 2 and the complex smart home application scenarios are abstracted, which will help developers to better understand and develop smart home scenarios applications.

Definition 1: The smart home application scenario is defined as:

$$SHSCENARIO = \bigcup_{i=1}^{\infty} SHScenario_i,$$
$$SHScenario_i =< USER, LOCATION, CONTEXT, DEVICE, FUNCTION, SERVICE > \quad (1)$$

Where $SHScenario_i$ denotes the smart home application scenario numbered i. $USER$ represents a collection of all users in the scenario, $LOCATION$ represents a collection of all locations in the scenario, $CONTEXT$ represents a collection of all locations in the scenario, $DEVICE$ represents a collection of all devices in the scenario, $FUNCTION$ represents a collection of all functions in the scenario, $SERVICE$ represents a collection of all services in the scenario.

Definition 2: The users in smart home application scenario is defined as:

$$USER = \bigcup_{i=1}^{\infty} User_i,$$
$$User_i =< id, u\_Name, LOCATION, des > \quad (2)$$

Where $User_i$ denotes the user in the smart home application scenario numbered i. id represents the user's unique identifier, u_Name represents the name of $User_i$,

$LOCATION$ represents a collection of all locations that the user is currently in, des is a set of descriptions of the user.

Definition 3: The locations in smart home application scenario is defined as:

$$LOCATION = \bigcup_{i=1}^{\infty} Location_i,$$
$$Location_i =< id, l\_Name, l\_Value, USER, DEVICE, CONTEXT, des > \quad (3)$$

Where $Location_i$ denotes the location in the smart home application scenario numbered i . id represents the location's unique identifier, l_Name represents the name of $Location_i$, $l_V$alue represents the value of $Location_i$, $l_V$alue is defined as a triple. $USER$ represents a collection of all users in the current location, $DEVICE$ represents a collection of all devices in the current location, $CONTEXT$ represents a collection of all contexts in the current location, des is a set of descriptions of the location.

Definition 4: The contexts in smart home application scenario is defined as:

$$CONTEXT = \bigcup_{i=1}^{\infty} Context_i,$$
$$Context_i =< id, c\_Name, c\_Value, LOCATION, SERVICE, des > \quad (4)$$

Where $Context_i$ denotes the context in the smart home application scenario numbered i . id represents the context's unique identifier, c_Name represents the name of $Context_i$, $c_V$alue represents the value of $Context_i$, $LOCATION$ represents the set of locations to which the current context belongs. $SERVICE$ represents a set of services that can monitor and control the context, des is a set of descriptions of the context, and establishes the relationship between the context and the service.

Definition 5: The devices in smart home application scenario is defined as:

$$DEVICE = \bigcup_{i=1}^{\infty} Device_i,$$
$$Device_i =< id, d\_Name, d\_Value, LOCATION, FUNCTION, des > \quad (5)$$

Where $Device_i$ denotes the device in the smart home application scenario numbered i . id represents the device's unique identifier, d_Name represents the name of $Device_i$, $LOCATION$ represents the set of locations to which the $Device_i$, belongs. $FUNCTION$ represents a set of functions of the $Device_i$, des describes all the device's function id and the relationship between the device and the function will be established through it.

Definition 6: The functions in smart home application scenario is defined as:

$$FUNCTION = \bigcup_{i=1}^{\infty} Function_i,$$
$$Function_i =< id, f\_Name, device.id, SERVICE, des > \quad (6)$$

Where $Function_i$ denotes the function in the smart home application scenario numbered i . id represents the function's unique identifier, f_Name represents the name of $Function_i$, $device.id$ indicates the id of the device to which $Function_i$ belongs. $SERVICE$ represents a set of services that the current function can provide, des represents a set of descriptions of functions. The relationship between the function and the service will be established through it and at the same time the relationship between the function and the device will be established.

Definition 7: The services in smart home application scenario is defined as:

$$SERVICE = \bigcup_{i=1}^{\infty} Service_i,$$
$$Service_i =< id, s\_Name, CONTEXT, FUNCTION, des > \quad (7)$$

Where $Service_i$ denotes the service in the smart home application scenario numbered i . id represents the service's unique identifier, s_Name represents the name of $Service_i$, $CONTEXT$ represents the set of contexts that the current service can monitor and control. $FUNCTION$ represents a collection of all functions that can provide the current service, des represents a set of descriptions of services. The relationship between the function and the service will be established through it and at the same time the relationship between the service and the context will be established.

We use OWL to build a conceptual model of smart home scenarios. Because the variant supported by the most popular tool (Protégé) [11] is OWL 2.0, we chose OWL 2.0. The conceptual model we define is the meta-metamodel of the smart home scenario, which is the basis for constructing the runtime knowledge graph.

B. The construction of the runtime knowledge graph

The types of devices in smart home scenarios and the programming interface are complex. We need a unified way to manage the underlying devices. We construct runtime models of smart devices in order to manage them in a unified manner. This is done easily with the help of SM@RT [12,13].

SM@RT (supporting models at run time) is a model-driven framework that supports model-based runtime system development. The process of construction of the runtime knowledge graph is as follows:

Step 1: Establish the objects and the relationship between objects according to the configuration file.

Step2：Use SM @ RT to get and update the device and user's runtime model information.

Step3：Update the relationship between objects through object information; Jump to step 2.

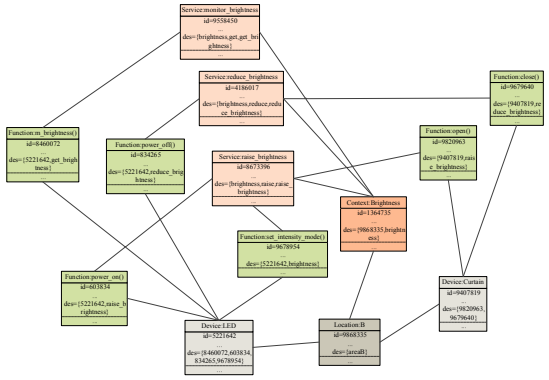

Figure 4. Establishment of the relationship between objects through the attribute des

As shown in Fig. 2, the device layer and the runtime knowledge graph layer are bidirectional. The synchronous engine not only enables any states of smart devices to be monitored by the runtime model, but also enables any changes to the runtime model to be applied on smart devices. Thus we can manage devices through operations on the runtime model, and these operations will propagate to underlying systems. The details of runtime model construction with SM@RT can be found in [14–16].

As shown in Fig. 4, there is a service (id=4186017) that can reduce light intensity, and its descriptive attribute is des= {brightness, reduce_brightness}. There is a context (id=4186017) about brightness and its descriptive attribute is des={9868335，brightness}. The same element {brightness} is in the des of both objects. Thus, we establish the relationship between the service (id=4186017) and the context (id=4186017).

Based on the device runtime model and the attribute des, we establish the relationship among user, location, context, service, function and device, and finally construct the runtime knowledge graph of the smart home scenario. The runtime knowledge graph layer can provide unified management for application development.

C. A code generation approach

Definition 8: The requirements in smart home application scenario is defined as:

$$REQUIREMENT = \bigcup_{i=1}^{\infty} REquirement_i,$$

$$REquirement_i =< rLocation, rContext\_x, xContextVR, \quad (8)$$

$$rContext\_y\_yContextVR\ level >$$

Where $REquirement_i$ denotes the requirement in the smart home application scenario numbered i. Each requirement can be managed by the scene developer. $rLocation$ represents the location to be controlled, $rContext\_x$ represents the context to be monitored, $xContextVR$ represents the range of the c_Value of $rContext\_x$. $rContext\_y$ represents the context to be controlled, $yContextVR$ represents the range of the c_Value of $rContext\_y$. $level$ indicates the $REquirement_i$ in the requirement engine priority.

The user can write the requirements profile as shown in Fig. 5. We parse the requirements profile and automatically generate the corresponding application code. The auto-generated code can be run directly on the previously-construct runtime knowledge graph, and the runtime model can be synchronized with the underlying device, and ultimately develop the application. Fig. 6 shows an automatically generated template code for a QVT modeling language.

According to the requirements profile, the code generator finds the context object needed to be controlled in the runtime knowledge graph and searches the set of functions that can be called through the runtime graph model. The parameters of the requirements files and the functions found by searching are filled in the template code, and finally the generated code can be used to develop smart home applications.

The way to write requirements files is close to people's logical way of thinking, and it's easy to write requirements files for domain experts or developers. The logic modeling program is automatically generated according to the demands of the users, and is applied to the runtime knowledge graph layer.

In our approach, domain experts or developers only need to focus on the requirements of the smart home scenarios, then the logical development of the application can be generated automatically.

```
<Requirment>
        <level  leveValue = {${x|x∈ {0,1,2,3,4,5,6,7,8,9}}}>
        </level>
        <rLocation  value = {${rlocation}}>
        </ rLocation >
        <rContext_x  des = {${rContext_x}}>
        </ rContext_x >
        <xContextVR  min = {${xContext.min}  max = {${xContext.max }}>
        </ xContextVR >
        <rContext_y  des = {${rContext_y}}>
         </ rContext_y >
        <yContextVR  min = {${yContext.min}}  max = {${yContext.max }}>
        </ yContextVR >
</ Requirment >
```

Figure 5. A sample template for requirements profile

```
…
mapping Requentment :: req(){
    var context_x = SHScenario.objectOfType(Context) -> select ( Context.des == ${rContext_x} and Context.location.des == ${rLocation} ) ;
    var context_y = SHScenario.objectOfType(Context) -> select ( Context.des == ${rContext_y} and Context.location.des == ${rLocation} ) ;
    …
        var serviceList = SHScenario.objectOfType(Service) -> select ( service.context == context_y  and service.dec == "set" ) ;
        while ( serviceList -> size() <> 0) {
            var service_z1 = serviceList -> selectOne(true);
            var functionList= SHScenario.objectOfType(Function) -> select ( function.service == service_z1 );
            while( functionList -> size() <> 0) {
                var function = functionList -> selectOne(true);
                function.execute();…}}}
    …
    if( context_x.c_value < xContextVR.min ) {
        var serviceList = SHScenario.objectOfType(Service) -> select ( service.context == context_y  and service.dec_s == "raise" ) ;
        while ( serviceList -> size() <> 0) {
            var service_x1 = serviceList -> selectOne(true);
            var functionList= SHScenario.objectOfType(Function) -> select ( function.service == service_x1 );
            while( functionList -> size() <> 0) {
                var function = functionList -> selectOne(true);
                function.execute();…}}}

    if( context_x.c_value > xContextVR.max ) {
        var serviceList = SHScenario.objectOfType(Service) -> select ( service.context == context_y  and service.dec_s == "reduce" ) ;
        while ( serviceList -> size() <> 0) {
            var service_x1 = serviceList -> selectOne(true);
            var functionList= SHScenario.objectOfType(Function) -> select ( function.service == service_x1 );
            while( functionList -> size() <> 0) {
                var function = functionList -> selectOne(true);
                function.execute();…}}}
    …
}
…
```

Figure 6. A QVT template for code automatically generated

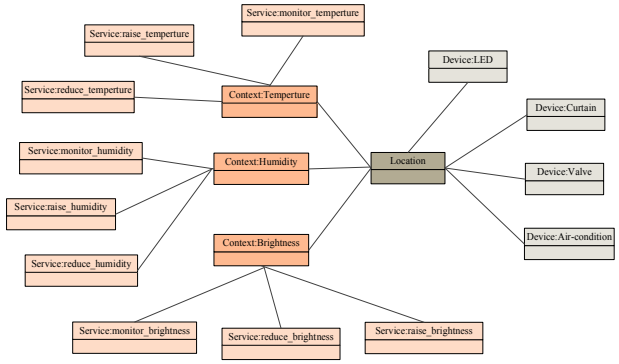

Figure 7. A user-defined meta-model of the smart planting scenario

V. CASE STUDY

Smart planting systems play an important role in the field of IoT. However, due to the diversity of smart devices and application scenarios, the cost of application development in the domain of smart planting is very high. In order to validate the feasibility and efficiency of our approach, we construct a smart planting system based on the runtime knowledge graph.

A. the meta-model of smart planting scenario

Taking the smart planting scenario in Fig. 1 as an example, we describe the scenario based on a conceptual model defined in Section 4.1. Fig. 7 shows the meta-model of that scenario. From Fig. 7 we can see there are four types of devices and three types of contexts defined in the current scenario, for each type of context there are some corresponding types of services. However, the instances of devices, functions, contexts, and services are not yet able to objectively reflect the current scenario in real time. We also need to incorporate the relationships between functions, devices and functions, and the relationships between functions and services into current knowledge by building a runtime model of the underlying device in the knowledge graph.

B. Construction of the runtime knowledge graphs

For a given meta-model and a given set of management interfaces, SM@RT [12] can automatically generate the code for mapping models to interfaces with sufficient runtime performance. In order to ensure that devices and functions in the runtime knowledge graph are correctly synchronized with the device runtime model, we define the mapping relationship between the models, and the runtime knowledge graph of area B is shown in Fig. 8.

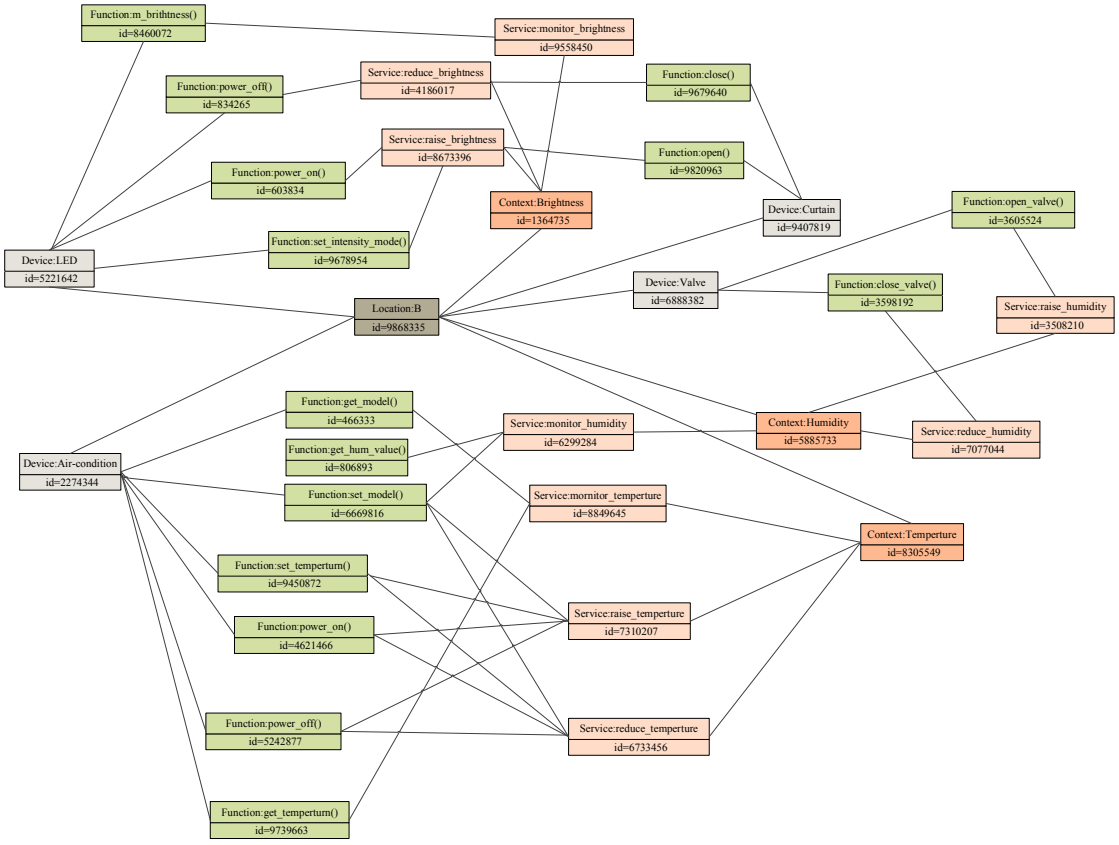

Figure 8. A runtime knowledge graph of area B

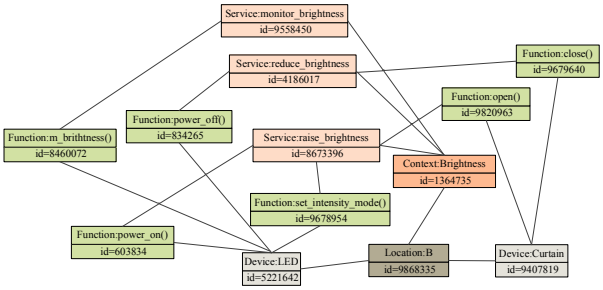

Figure 9. The search result of $req_{monstera,brightness}$

C. code automatically generated

For example, monstera is planted in area B, and the need for light intensity when breeding is like follow:

$$req_{monstera,\ \text{brightness}} = < \{areaB\}, \{brightness\}, \{50, 60\},$$
$$\{brightness\}, \{50, 60\}, 4 >$$

According to the req, we search in Fig. 8, then the result shown in Fig. 9 .This process is as follows:

Step 1: Search location where location.des={areaB}, then get the location (id=5221642) as return.

Step 2: Search location.context where and context.des= {brightness},then get the context(id=1364735) as return.

Step 3: Search context.service where service.des = {raise}, get the service(id=8763396) as return.

Step 4: Search service.function, get two functions (id=603834, id=9820963) as return.

According to the QVT template in Fig. 6, an application code is generated to act on the runtime knowledge graph in Fig. 8. Finally, when the light intensity is less than 50, the smart LED lamp is turned on and the curtains are opened to increase Light intensity.

D. Evaluation

We evaluate our approach from three aspects.

1) Construction of IoT systems for smart system based on runtime knowledge graph

For constructing smart device runtime knowledge graph, developers only need to define the relations among the objects, define the architecture-based meta-models and the access models on the eclipse modeling framework (EMF). All the application logic can be carried out through executing operating programs on the runtime knowledge graph without handling the different types of low-level management interface. Therefore, the extra work to construct t the runtime knowledge graph is acceptable for smart home application development.

We have developed smart home systems of smart planting system based on the runtime knowledge graph as shown in Fig. 8. Every model element in the runtime knowledge graph represents an entity in the real world. Administrators can manage smart devices at an architecture level and the operations on the runtime knowledge graph are transformed into invocations of management interfaces of the underlying system. Furthermore, all the application logic can be carried out by executing programs on the runtime knowledge graph, this may benefit from many model-centric analysis or planning methods and mechanisms such as model checkers.

The process of constructing our knowledge graph at run time by our approach is one-off, and this time-cost is acceptable. The runtime knowledge graph can be reused in different smart home application scenarios.

2) Comparison of the programming difficulty of using management interfaces and using runtime knowledge graphs

The architecture-based runtime model models the underlying APIs, reusing the data access code. Then, developers can focus on the management task logic without handling different types of low-level management interface.

TABLE 1. COMPARISON OF LOC AMONG DIFFIERENT APPROACHES FOR APPLICATION DEVELOPMENT

Development approach	LOC(Line of code)
$profile_{\text{manual-programming}}$	6
$QVT_{\text{automatic-generated}}$	25
$Java_{\text{automatic-generated}}$	212
$QVT_{\text{manual-programming}}$	18
$Java_{\text{manual-programming}}$	144

Take an example of one requirement about brightness of the smart planting system. As shown in Tab.1, the profile code is of the smart brightness system is just 6 LOC, the QVT code generated is about 25 LOC, the Java code generated is about 212 LOC. In contrast, the QVT code written manually is 18 LOC, the Java code written manually is 144 LOC. Although the generated QVT code more than 40% of the QVT code written manually and the generated Java code more than 47% of the QVT code written manually, but in general, our approach can reduce LOC by about 85 percent at least.

In addition, the requirements of smart home scenarios are complex and changeable. Our approach enables the management of requirements. Due to the low threshold of management requirements, developers or experts can manage the application configuration files to achieve the development, while greatly saving the time to develop applications.

3) Comparison of the response time

Take an example of the smart planting system. As shown in Tab.2, the average response time for the general approach is 1462 milliseconds, and the average response time for our approach is 1806 milliseconds.

TABLE 2. COMPARISON OF THE RESPONSE TIME BETWEEN TWO APPROACHES FOR SMART PLANTING SYSTEM DEVELOPMENT

Development approach	Average response time
general approach	1462ms
our approach	1806ms

Our approach maintains a runtime knowledge graph during operation, and the values of context are updated periodically. When running the logic model, we can directly read the values in the knowledge graph, and do not need to call the corresponding monitoring functions like the general approach. However, our approach is still slower in response time. The main reason is that the two sets of programs are based on the same APIs and there are some extra operations in the runtime model based approach, which are aimed to ensure synchronization between the architecture-based runtime model and the underlying system. However, the difference is small and completely acceptable for IoT system management.

VI. Conclusion and Future Work

Smart home application development usually requires developers to deal with low-level system problems, understand the scenario and handle complex and changing needs. This paper proposes a runtime knowledge graph based approach to smart home application development. Developers only need to define the meta-model based on the conceptual model and construct runtime knowledge graphs of smart devices. Developers can manage the requirements of the scenarios. Requirements files automatically generate modeling languages that run directly on the runtime knowledge graph. The code on the runtime knowledge graphs are automatically mapped to smart device and all the application logic can be carried out by executing the generated code on the runtime knowledge graphs. Thus, developers can focus on the core of management logic. Our approach can help developers handle underlying APIs, complex application logic and the collected low-level data, which greatly reduces the workload of hand coding.

As future work, we plan to improve support for developers to program smart devices. We plan to perform further analysis such as model checking to ensure a deeper correctness and completeness of the generated causal link between the runtime knowledge graph and underlying systems. And also, we plan to add more advanced functions with the help of model techniques to ease development tasks of IoT systems.

Acknowledgment

This work was supported by the National Key R&D Program of China under Grant No. 2017YFB1002000, the Guiding Project of Fujian Province under Grant No. 2018H0017, the Fujian Collaborative Innovation Center for Big Data Applications in Governments.

References

[1] D. C. Schmidt, "Model-Driven Engineering," Computer Languages Systems and Structures vol. 43, 2006, pp. 139-155.

[2] S. Sicard, F. Boye and N. D. Palma, "Using components for architecturebased management: the self-repair case," International Conference on Software Engineering, ACM, May. 2008, pp. 101-110, doi: 10.1145/1368088.1368103.

[3] B. Morin, O. Barais, G. Nain and J. Jezequel, "Taming dynamically adaptive systems using models and aspects," IEEE International Conference on Software Engineering, IEEE, May. 2009, pp. 122-132, doi: 10.1109/ICSE.2009.5070514.

[4] J. Yang, G. Huang, W. H. Zhu, X. F. Cui and H. Mei, "Quality attribute trade-o through adaptive architectures at runtime," Journal of Systems and Software, vol. 82, 2009, pp. 319-332, doi: 10.1016/j.jss.2008.06.039.

[5] H. Mei, G. Huang, L. Lan and J. G. Li, "A software architecture centric selfadaptation approach for internet-ware," Science in China, vol. 51, 2008, pp. 722–742, ISSN: 1009-2757.

[6] S. Nakamura, S. Shigaki, A. Hiromori, H. Yamaguchi and T. Higashino, "A model-based approach to support smart and social home living," ACM International Joint Conference on Pervasive and Ubiquitous Computing, ACM, Sept. 2015, pp. 1101-1105, doi: 10.1145/2750858.2805835.

[7] D. Lüddecke, N. Bergmann and I. Schaefer, "Ontology-Based Modeling of Context-Aware Systems," Model-Driven Engineering Languages and Systems, Springer International Publishing, LNCS, vol. 8767, 2014, pp.484-500, doi: 10.1007/978-3-319-11653-2_30.

[8] D. Ernadote, "Ontology-Based Pattern for System Engineering," 2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS). IEEE, Sept. 2017, pp. 248-258. doi:10.1109/MODELS.2017.4.

[9] D. Ernadote, "An ontology mindset for system engineering," IEEE International Symposium on Systems Engineering, IEEE, Sept. 2015, pp. 454-460, doi: 10.1109/SysEng.2015.7302797.

[10] B. Iordanov, A. Alexandrova, S. Abbas, T. Hilpold and P. Upadrasta, "The Semantic Web as a Software Modeling Tool: An Application to Citizen Relationship Management," Model-Driven Engineering Languages and Systems, Springer Berlin Heidelberg, LNCS, vol. 8107, 2013, pp. 589-603, doi: 10.1007/978-3-642-41533-3_36.

[11] X. Chen, A. Li, X. Zeng, W. Guo and G. Huang, "Runtime model based approach to iot application development," Frontiers of Computer Science, vol. 4, 2015, pp. 540-553, doi: 10.1007/s11704-015-4362-0.

[12] B. Motik, "Representing and Querying Validity Time in RDF and OWL: A Logic-Based Approach," International Semantic Web Conference, Springer Berlin Heidelberg, LNCS, vol. 6496, 2013, pp. 550-565, doi: 10.1007/978-3-642-17746-0_35.

[13] H. Song, et al, "Supporting runtime software architecture: a bidirectional-transformation-based approach," Journal of Systems and Software, vol. 84, May. 2011, pp. 711-723, doi: 10.1016/j.jss.2010.12.009.

[14] J. Li, X. Chen, G. Huang, H. Mei and Chauvel, F, "Selecting fault tolerant styles for third-party components with model checking support," Lecture Notes in Computer Science, vol. 5582, 2009, pp. 69-86, doi: 10.1007/978-3-642-02414-6_5.

[15] X. Chen, G. Huang, F. Chauvel, Y. Sun and H. Mei, "A framework for the integration of MOF-compliant analysis methods," Asia-Pacific Symposium on Internetware, ACM, Nov. 2010, pp. 1, doi: 10.1145/2020723.2020724.

[16] O. M. Group, Meta object facility (mof) 2.0 core specification. http://www.omg.org/cgi-bin/apps/doc?ptc/04-10-15.pdf, 2013.