A Formal Methodology for Easing Development and Maintenance of Entity Services in Service Oriented Software-Defined Internet of Things

Haiming Chen, Member, IEEE, Kaibin Xie, Li Cui, Member, IEEE, and Antonio Pescapè, Senior Member, IEEE

Abstract—Internet of Things (IoT) systems are usually built with entity services, which are those abstracting functionalities of sensing and executing devices in the physical space. As requirements of sensing or controlling the physical space can be varied with different systems, entity services are supposed to be easily adapted to meet such dynamicity. To ease updating and modification of entity services, although a software-defined network approach has been applied in building IoT systems, entity services developed with the same software architecture as traditional services on the Internet have an inherited problem in adaptability. In order to solve the problem, we abstract the functionalities of an entity service in social, cyber, and physical spaces into application model, sense-execute model and physical model respectively, and propose a Physical Model Driven software Architecture (PMDA) for guiding design of entity services. To ease development of entity services, we also propose a formal Development method of Entity Services (fMES) to transform the abstracted models of PMDA into implementable software modules. Besides, to reduce maintenance cost of entity services when adapting them to different requirements from the social space, we propose a formal Maintenance method of Entity Service (fMES). The correctness of fDES and fMES is verified by a case study, and their effectiveness in reducing cost of developing and maintaining IoT systems comprised of large-scale frequently-changed entity services is proved by analysis.

Index Terms—Internet of Things, entity service, edge service, software architecture, evolution mechanism.

I. INTRODUCTION

With the rapid development of smart sensing technology, wireless networking technology, and embedded computing technology, more and more digital devices deployed in the physical world can be interconnected and interoperated to compose a new cyber-physical-social environment [1], named Internet of Things (IoT) [2]. The goal of IoT is to connect any people and any things, in any places, at any time, in the form of services [3], [4], which can be mainly categorized into two types, namely application services and entity services.

H. Chen is with the Faculty of Electrical Engineering and Computer Science & Zhejiang Provincial Key Laboratory of Mobile Network Application Technology, Ningbo University, Zhejiang, China, 315211. E-mail: chenhaiming@nbu.edu.cn
K. Xie and L. Cui are with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, 100190. Email: lcui@ict.ac.cn
A. Pescapè is with the Department of Electrical Engineering and Information Technology, University of Napoli Federico II, Napoli, Italy, 80125. Email: pescape@unina.it
Copyright © 2019 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending a request to pubs-permissions@ieee.org.

The application services are those used for processing data [5], while the entity services are those for abstracting functionalities of sensing devices or smart gateways. As shown in Fig. 1, application services are usually built at the cloud side [6], while entity services can be built at the cloud side and the device side (a.k.a. edge services [7] or fog services [8]). These services are seen as ubiquitous ingredients to construct IoT systems in every walk of life. According to the ternary theory [9], entity services consists of parts in three spaces, namely social space, cyber space and physical space. The ternary characteristic of entity services make them different from traditional application services on the Internet. In particular, entity services in the cyber space interact with people in the social space, who send requirements to devices in the physical space. The people’s requirements are usually sensing data from the physical space and generating control information to change the status of devices in the physical space [10].

Because requirements for sensing and controlling the physical space can be varied from person to person and over time, the physical part of entity service can be changed frequently. To ease updating of entity services, a software-defined network architecture has been applied in building IoT system, which are called Software-defined IoT [11], [12]. In particular, there is a central point (i.e. the software-defined IoT controller) serving as the control plane of the IoT system, and the entity services at both the device and cloud sides are treated as the data plane to be developed and maintained. Currently, there has been some work done on the design of the control plane of the service-oriented software-defined IoT [13], [14]. However, the data plane of the service-oriented software-defined IoT [15] has hardly been changed from distributed objects (e.g. Physicalnet [16]), software agents (e.g. ASO [17]), or web services (e.g. SODA [18] and TinyREST [19]).
Because they all encapsulate device- and scenario-specific functionalities into logic components in the same manner as busi
ness process modules, it is difficult to adapt them to the dynamic requirements from the social space and to change parameters of sensing or controlling the physical space.

To address the problem, some approaches have been proposed, such as supplying adaptation rules [20], [21] or self-adaptive methods [22] for the existing software architecture. However, the proposed adaptation solutions essentially do not change the software architectures designed for traditional Internet software [23]. So the developed IoT systems with these solutions inherit behavior characteristics from the traditional Internet software, especially deficiency in interacting with the social space and the physical space simultaneously. Therefore, it still implies lack of efficiency to interoperate with the control plane, especially the entity service maintenance and deployment module (ESMDM), to update entity services. So it is necessary to build a new design model of entity service to support cost-effective modification of the sensing or controlling parameters of entity services in the physical space to fulfill dynamic requirements from the social space.

In this paper, based on our previously designed software architecture for developing IoT application systems [24]–[27], we design a Physical Model Driven software Architecture (PMDA) and a formal Development method of Entity Services (fDES) for guiding implementation of entity services in the data plane of software-defined IoT. PMDA considers common characteristics of entity services according to the ternary theory, and decomposes the functionalities of an entity service in social, cyber, and physical spaces into three models, namely application model, sense-execute model and physical model. fDES transforms the abstracted models of PMDA into implementable software modules to ease development of entity services. Correspondingly, to reduce maintenance cost of entity services when adapting them to dynamic requirements from the social space and changing monitoring parameters of the physical space, we propose a formal Maintenance method of Entity Service (fMES) to guide implementation of the ESMDM of the control plane of software-defined IoT. The correctness of fDES and fMES is verified by a case study. The effectiveness of fDES and fMES in reducing cost of developing and maintaining IoT application systems comprised of large-scale frequently-changed entity services is proved by analysis.

The main contributions of the paper are embodied in the following aspects.

1. We propose a Physical Model Driven software Architecture (PMDA) for guiding design of entity services in the data plane of software-defined IoT. Based on PMDA, we propose a formal Development method of Entity Services (fDES) to transform the abstracted models of PMDA into implementable software modules to ease development of entity services.

2. Considering dynamic requirements from the social space, we design a formal Maintenance method of Entity Service (fMES) to reduce complexity and difficulty of maintaining entity services.

3. Some entity services based on fDES and the main processes of fMES are implemented in JCSP [28]. Results of a case study show that our proposed methods are correct, and analytical results prove that they are more effective in reducing the cost of developing and maintaining IoT application systems, which are composed of large-scale entity services, especially with frequent changes of requirements.

The remainder of this article is organized as follows. Section II describes the related work on developing and maintaining software systems with dynamic requirements. Brief introduction of PMDA and detail on fDES are presented in Section III. Detail on fMES is presented in Section IV. Section V shows the case study of developing and maintaining entity services in an IoT system with the methods proposed in this paper. In Section VI, we make a concluding remark and bring forth our future work. Some abbreviations used in this paper are listed in Table I.

## II. RELATED WORK

### A. Software architectures for developing entity service

At present, entity services mostly developed with the same software architecture of inter-connectable computing module in the Internet environment. Particularly, the existing software architectures for developing entity service can be categorized into following classes.

1. Distributed object, which uses IDL (Interface Definition Language) to describe services provided by entity services. Interconnection among them is based on the RPC (Remote Procedure Call) or RMI (Remote Method Invocation) protocol. Data representation format of exchanged message is based on CDR (Common Data Representation). Physicalnet [16] is such a type of entity service.

2. Software agent, which can be built with software architecture similar to distributed object, while integrating smart capabilities, such as knowledge-based reasoning and context-aware adaptation [29], into entity services. They usually adopt ACL (Agent Communication Language) and MTP (Message Transport Protocol) to exchange message. ASO [17] is known as such a kind of entity service.

3. SOAP-based web service, which is built on the HTTP protocol, and takes WSDL (Web Service Description Language) to describe services provided by entity services. Accordingly, some mechanisms for register and discover services should be implemented. SODA [18], DPWS [30] and SensorWeb [31] are designed for developing web services on embedded devices.
D. SDD adopts update methods to change the attributes of Cyber-D and add update rules about Cyber-D. However, none of the existing work has given formal results on the design of entity service in IoT and its evolution mechanism. In this paper, we develop an evolution mechanism based on a new software architecture proposed for developing entity service, which has extracted common factors of different IoT applications. It is worth pointing out that in our previous work [24]–[27], we have proposed some approaches to developing and maintaining IoT systems more easily. However, none of those approaches has been presented from the aspect of entity service so completely as in this paper, which includes the design of software architecture, the formal development method, and the formal maintenance method.

III. FORMAL DEVELOPMENT METHOD OF ENTITY SERVICE

A. Software Architecture of Entity Service

PMDA (Physical Model Driven Architecture) is a software architecture to guide the development of entity services, which can be interconnected to build IoT application systems. It considers common characteristics of entity services according to the ternary theory, and separates the functionalities of an entity service in social, cyber, and physical spaces from each other, and abstract them into three models, which are application model, sense-execute model and physical model, as shown in Fig. 2. The application model delivers the requirement information (req-info) from the social space. The sense-execute model receives req-info from the application model, and processes sensory data (sen-data) from the physical space and generates execution information (exe-info) to the change the status of physical space, according to the req-info and the sen-data. The physical model provides sen-data to the sense-execute model and receives exe-info from the sense-execute model to interact with the physical space.

Referring to the whole structure of PMDA, we can see that entity services in IoT application systems can parse the requirements of users, and take proper action to sense or change the status of the physical entities. To apply the software architecture in implementing IoT application systems, we refine all the models of PMDA, as shown in Fig. 3. The Application Model has two components, namely REQ and EXTR. The component REQ delivers the requirements from the users, and the component EXTR extracts the requirements which mostly include operations on the physical environment.

The Sense-Execute Model has six components, which are JUDGE, ASSOCIATE, DECOMPOSE, SENSE, PROCESS and EXECUTE. The component JUDGE decides whether the requirements can be fulfilled without involving other entity services besides itself. If false, the component ASSOCIATE

![Diagram](image-url)
processes the requirements and forwards the requirements to other entity services. If true, the component DECOMPOSE decomposes the requirements into two parts, which are related to sensing and controlling operations on the physical environment respectively. Accordingly, the component SENSE collects the required sensory data, and the component EXECUTE generates control information based on the requirements for control and the sensing information. The component PROCESS, which is between SENSE and EXECUTE, is responsible for processing the sensory data and generates the sensing information.

The Physical Model has two components, which are named OBJECT and ACT respectively. The component OBJECT provides the sensory data to the component SENSE. The component ACT acts on the component OBJECT and changes status of the physical environment according to the control information of the component EXECUTE.

The composition of the three models in PMDA and the interactions among the models are formally described by an ADL (Architecture Description Language) named Wright [44]. Effectiveness of constructing entity services based on PMDA has been verified by PAT (Process Analytical Toolkit) [45], in terms of deadlock-free, divergence-free and non-terminating. Besides that, the same properties of IoT application systems developed under guidance of PMDA have been proven by mathematical induction [24].

In the following subsections, we design a formal method for developing entity services based on PMDA, which is named fDES. It includes the following two steps. Firstly, based on the principle of CSP (Communicating Sequential Processes) [46], we transform the components of PMDA into software modules. Secondly, we analyze the interaction patterns of these software modules and connect these software modules together to form an entity service.

B. Transform the components of PDMA into software modules

According to above description of PMDA, an entity service is composed of three models, each of which is refined into more detailed components. Since CSP is widely adopted to formally define composition and interaction pattern of software systems, we use CSP to transform the abstract components of PMDA into concrete software modules. According to CSP, components of PMDA can be transformed into corresponding elements, which are named components, processes, and events. The meanings of the three elements are as follows. An event denotes an action of a behavior. A process denotes one behavior of a task. A component denotes a specific task of an entity service. The structural relationship of the three elements is as follows. The event is the basic element for composing the process and the component. A process is a sequence of events. A component is composed of several processes. All the processes of a component can be categorized into different groups. The processes in the same group are executed in sequence. The processes in different groups are executed in parallel.

Software modules correspond to the three CSP elements are named as Module-E, Module-P and Module-C respectively. A Module-E realizes the functions of an event; a Module-P realizes the functions of a process and a Module-C realizes the functions of a component. According to the structural relationship of the three elements mentioned above, a Module-P contains several Module-Es, and a Module-C is composed of several Module-Ps. In order to connect Module-Es in the Module-P or Module-C, we use Interface-R to receive the related information from other Module-Es and Interface-S to send the generated information to the related Module-Es. The symbols of Module-E, Module-P, Module-C, Interface-R and Interface-S are depicted in Fig. 4. The structural relationship of Module-E, Module-P and Module-C and their interfaces are depicted in Fig. 5.

C. Analyze interaction patterns of Module-Es in Module-P

In Fig. 5, we have shown an interaction situation of Module-Es. More generally, there are five situations of interaction among Module-Es. (1) Two Module-Es interact in the same Module-P. (2) Two Module-Es interact between two Module-Ps of the same Module-C. (3) A Module-E of one Module-
D. Connect software modules to form an entity service

We describe this step of fDES by an example of developing an entity service for environment monitoring. The goal of the entity service is built on an air conditioner to monitor the temperature and the humidity of a room and make people feel comfortable in the room. We consider that environment with temperature from 23°C to 27°C and humidity from 40% to 60% is comfortable.

Firstly, we develop ten Module-Cs, which are REQ-C, EXTR-C, JUDGE-C, ASSOCIATE-C, DECOMPOSE-C, SENSE-C, PROCESS-C, EXECUTE-C, OBJECT-C and ACTUATE-C, in accordance with the ten components of PMDA. Each of them is composed of Module-Ps. Fig. 7 depicts all the Module-Cs and their interactions for developing the entity service.

(1) The application model of the entity service consists of two Module-Cs, namely REQ-C and EXTR-C. Here we take REQ-C as an example to show the rule of Module-P in comprising Module-C. REQ-C processes the requirements (req) of users in the room, and has only one Module-P, named REQ-P, which implements the task of REQ-C. REQ-P has three Module-Es, named RECV-E, PROC-E and SEND-E. Due to space limitation, we don’t show the Module-Es in each Module-P.

(2) The sense-execute model of the entity service consists of six Module-Cs, namely JUDGE-C, ASSOCIATE-C, DECOMPOSE-C, SENSE-C, PROCESS-C, and EXECUTE-C. JUDGE-C judges whether req-info can be fulfilled in the current entity service, and send req-info to ASSOCIATE-C or DECOMPOSE-C accordingly. ASSOCIATE-C searches the related entity services according to req-info, and forwards req-info to the related entity service. DECOMPOSE-C is to decompose req-info into two parts, which are sensing or controlling the environment condition of the room, denoted as req-o and req-c respectively. SENSE-C collects the sensory data (i.e. temperature and humidity readings) of the room and sends these sensory data to PROCESS-C. PROCESS-C processes the sensory data and sends the generated sensory information to EXECUTE-C. EXECUTE-C interacts with the ACT-C which is in the physical space, and generates the execution command according to req-c and the sensory information.

(3) The physical model of the entity service is composed of OBJECT-C and ACT-C. OBJECT-C provides the sensory

---

Fig. 6. (a) ONE-TO-ONE interaction pattern. (b)ONE-TO-MANY interaction pattern. (c)MANY-TO-ONE interaction pattern.

---

Fig. 7. Module-Cs and Module-Ps and their interactions for developing an environment monitoring entity service.
data (i.e. temperature and humidity readings). ACT-C receives the control command from EXECUTE-C and triggers related actuators according to the control command.

E. Analysis of the advantages of fDES

To show the advantages of developing entity services with fDES, we prove that the fDES satisfies three principles of loosely coupled design.

(1) fDES satisfies the Dependency Inversion Principle (DIP) [47]. The central point of DIP is that the high-level module should not depend on the low-level module, while both should depend on the abstraction. According to fDES, the high-level modules can be regarded as the Module-Es, which receives information from other Module-Es. The low-level modules can be regarded as the Module-Es, which send information to the related Module-Es. In an entity service, the high-level module interacts with the low-level module only through the Interface-R and the Interface-S. That is to say, the two modules depend on the related event, while the high-level module is independent of the low-level module. So fDES satisfies the DIP, which means the high-level Module-E is independent of the low-level Module-E in an entity service.

(2) fDES satisfies the Law of Demeter (LoD) [48]. LoD requires that each module in a software system should have the least knowledge of the other modules. That is to say, a module should not interact with the other modules if it is not necessary. According to fDES, each Module-E only interacts with the other Module-Es when they must interact to finish the behavior of a task or to transmit necessary messages to other Module-Es. We can see that each Module-E only has the necessary information related to the other Module-Es. That is to say, each Module-E has the least knowledge of the other Module-Es. So fDES satisfies the LoD, which means all the Module-Es are low-coupling.

(3) fDES satisfies the Single Responsibility Principle (SRP) [47]. SRP requires that each module of a software system should have only one responsibility. According to fDES, each Module-E represents only one action of a behavior, e.g. RECV-E in REQ-P only represents the action of receiving requirement from the social space, while PROC-E and SEND-E represent the action of processing and sending the requirement respectively. An action of a behavior can be regarded as a responsibility of a module. So fDES satisfies the SRP, which means all the Module-Es are high-cohesion.

According to the above analysis, we can see that each Module-E in an entity service has three properties, which are high-independence, high-cohesion and low-coupling. Entity services with these three properties are seen as loosely coupled and easily maintainable.

IV. FORMAL MAINTENANCE METHOD OF ENTITY SERVICE

Considering the dynamic requirements from the social space, the sensing or controlling parameters of the related entity services in the physical space should be changed frequently. So we first analyze the types of changing physical models, and figure out corresponding operations for changing the sense-execute models. Then we present the formal Maintenance method of Entity Service (fMES).

A. Types of changing physical models

The two key characteristics of physical models are physical area and physical parameters, which can be different for different IoT applications systems. So we can represent physical models with these two characteristics, as illustrated in Fig. 8, where pmm is short for the name of physical model, par is short for physical area, and pps is short for physical parameters. The possible changing types of physical models are determined by both changing physical area and changing physical parameters.

<table>
<thead>
<tr>
<th>name of physical model (pmm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>physical area (par)</td>
</tr>
<tr>
<td>physical parameters (pps)</td>
</tr>
</tbody>
</table>

Fig. 8. Representation of a physical model.

1) Changing physical area: We denote the physical area of a physical model as a circle (x, r), which is widely taken as a simplified theoretical sensing coverage model of smart devices [49], [50]. x is the center of the physical area, and r is the radius of the area. So for the physical model in an entity service, its physical area can be changed as follows.

(1) Shrink the location (SHRINK). For example, originally an entity service can monitor an area with radius of r meters, but now it can only monitor an area with radius of s meters (s < r).

(2) Enlarge the location (ENLARGE). It can be explained with the same example as shown above, but now it can monitor an area with radius of e meters (e > r).

(3) Move to new location (MOVE). For example, the center of the physical area of the entity service (i.e. x = B) is moved to a new place D (i.e. x = D).

2) Changing physical parameters: For the physical model in an entity service, its physical parameters can be changed as follows.

(1) Add new physical parameters (ADD). For example, originally an entity service can monitor CO and CO₂. If the node is added a pH sensor, the physical model should be extended to provide the new function.

(2) Delete some physical parameters (DELETE). For example, the entity service stops providing the CO₂ parameter for IoT applications.

(3) Replace physical parameters (REPLACE). For example, the entity service stops providing a part of (or all of) physical parameters (i.e. CO₂), and adds some new sensors (i.e. pH).

3) Changing both physical area and physical parameter: Based on the three changing types of par and three changing types of pps for a physical model, we can deduce that there are nine types of changing both par and pps of a physical model. The nine changing types are illustrated as follows. (SHRINK, ADD), (SHRINK, DELETE), (SHRINK, REPLACE), (ENLARGE, ADD), (ENLARGE, DELETE), (ENLARGE, RE-
changing types for the physical models are (*, DELETE), (MOVE, ADD), (MOVE, DELETE), (MOVE, REPLACE).

It is worth noting that when the radius of physical area is shrunken to zero or its center is changed to null, or the last physical parameter is deleted, the entity service is removed from the physical space or all its functions is terminated. In other words, an extreme case of the changing types (SHRINK, DELETE) and (MOVE, DELETE) occurs, when either par or pps is changed to null.

B. Operations for changing sense-execute models

Based on the above analysis of types of changing physical models, in this section we figure out operations for changing the sense-execute models accordingly. As shown in Fig. 7, a sense-execute model interacts with a physical model to process physical parameters of an entity service. Representation of a sense-execute model is illustrated in Fig. 9. The name of the sense-execute model is denoted as semn. The process ability of a sense-execute model is denoted by pro, which consists of several physical parameters it can process. Pairs of <par, pps> of related physical models are listed below the pro of the sense-execute model. The relationship between a sense-execute model and physical models is illustrated in Fig. 10. The sense-execute model named moniEnviron has process ability pro = CO + CO₂ + Humidity + pH + O₂, which means it can process any one of these five physical parameters, namely CO, CO₂, Humidity, pH and O₂, and any combination of them. More than one physical models (i.e. pmB, pmW, and pmZ) can share a sense-execute model. When physical models change, the related sense-execute model should undergo some operations to change accordingly. Through analysis, we find that the operations include ERASE, UPDATE and LOOKUP, which are introduced as follows.

1) The UPDATE operation: For the DELETE changing type of physical parameters, the changed physical parameters (pps) of the physical models are still subsets of the process ability (pro) of the related sense-execute model. So if the changing types for the physical models are (*, DELETE), where * means the types of changing physical area can be any of SHRINK, ENLARGE, and MOVE, the deleted items of <par, pps> of the related sense-execute model should be updated, while the relationship between the changed physical model and the sense-execute model does not need to be updated.

2) The LOOKUP operation: If the changing types for the physical models are (*, ADD), (*, REPLACE), and the changed pps of the physical model is not in the process ability (pro) of the corresponding physical processing system, a LOOKUP operation is needed to setup a new relationship between the physical model and the found sense-execute model. The LOOKUP operation mainly consists of following steps: (1) remove the related <par, pps> from the sense-execute model; (2) remove the relationship between the physical model and the sense-execute model; (3) search an appropriate sense-execute model for the changed physical model according to the new physical parameters.

3) The DEPLOY operation: If no appropriate sense-execute model is found for the new physical model, a new sense-execute model should be deployed. The DEPLOY operation includes two steps. The first step is to register the physical area and physical parameter of the new physical model to the deployed sense-execute model. The second step is to setup a relationship between the deployed sense-execute model and the new physical model.

4) The ERASE operation: If the changing type of physical model is ZERO, which means the physical model terminates because its physical area or physical parameter becomes null, the related <par, pps> should be removed from the corresponding sense-execute model and the relationship between the sense-execute model and the physical model should also be removed.

5) The UPDIRS operation: It is worth noting that in the above analysis of types of changing physical model and operations for changing sense-execute model, we can see that for the operations of UPDATE, we should update the items of <par, pps> of the related sense-execute model. For LOOKUP, DEPLOY and ERASE, we should not only update the items of <par, pps> of the related sense-execute model, but also update (i.e. remove or setup) the relationship between the sense-execute model and physical models. So we define the operation of updating the relationship between physical models and the related sense-execute model as UPDIRS.

C. Overview of fMES

Based on the above analysis of types of changing physical models, and corresponding operations for changing the sense-execute models, we design the formal maintenance method of entity service (fMES) with the following procedures. (1) Determine the initial relationship between the physical models and the sense-execute models; (2) Judge the changing types of the physical models; (3) Update the sense-execute models in accordance with the changed physical model. At the same time, change the interaction relationship between the changed physical model and the updated sense-execute model. If the required sense-execute models don’t exist, deploy them.
In the following subsections, we first take the DEPLOY operation as an example to illustrate the procedures to maintain an entity service, and then present its formal description.

D. Illustration of fMES

Take the environment monitoring entity service shown in Fig. 7 as an example, and suppose that users want to monitor the PM2.5, besides temperature and humidity of the room, so the developer should add a new component to sense the concentration values of the PM2.5 and make an air purifier work when it is higher than 80. Based on the entity service model developed by fDES, which is shown in Fig. 7, and the description on operations for changing sense-execute models in the previous subsections, we know that a DEPLOY operation should be done to fulfill the requirement of changing physical parameter. The whole procedure to change the physical model and the related sense-execute model is described as follows, and the entity service after changing operation is depicted in Fig. 11.

(1) Add the software modules OBJECT2.5-C and ACT2.5-C to physical model for PM2.5 monitoring and air purifying.

(2) Lookup an appropriate sense-execute model for PM2.5 monitoring and air purifying. Assuming that it is not found, so the developer deploys a new sense-execute model, to add some Module-Ps to SENSE-C, PROCESS-C and EXECUTE-C in the sense-execute model of the entity service.

For the SENSE-C, a new Module-P named DIVSEN-P is added to divide the requirements in the RECR-P into two parts. So two new modules named CONTEM-P and CON2.5-P are added for controlling temperature/humidity and PM2.5 respectively. Besides, three new modules named CON2.5-P, REC2.5-P and EXE2.5-P, are added for receiving the sensory information of PM2.5, generating the control command of the air purifier.

(3) Change some modules in JUDGE-C. In particular, the existing modules LOC-P and EXT-P are replaced with LOC2.5-P and EXT2.5-P respectively, to handle environment monitoring requirements included with PM2.5 besides temperature and humidity. We use the JUD2.5-P to interact with the ASSOCIATE-C instead of the EXT2.5. Correspondingly, the modules in ASSOCIATE-C and DECOMPOSE-C are changed to receive requirements from EXT2.5-P and LOC2.5-P of the JUDGE-C instead of EXT-P and LOC-P.

(4) The modules REQ-C and EXTR-C in the application model of the environment monitoring entity service does not need to change, because the way to process the changing requirements from the social space can be the same, even though it can differ for the changing parameters of sensing or controlling the physical space to induce the evolution of the environment monitoring entity service.

E. Formal description of fMES

According to the above explanation of types of changing physical model and all possible operations for changing corresponding sense-execute model, we express the evolution rule of entity services by

\[ \{R_{\text{sem.pm}}, (C_{pm}, pm'_{j})\} \rightarrow \{O_{\text{sem.pm}}, R_{\text{sem.pm}}\}, \]

where \( R_{\text{sem.pm}} \) is the set of relationship between sense-execute model and physical model, \( (C_{pm}, pm'_{j}) \) represents the type of changing physical model and the changed physical model respectively, \( O_{\text{sem.pm}} \) is the operation for changing the sense-execute models and the relationship.

\[ R_{\text{sem.pm}} = \{(sem_1, pm_{i1}), (sem_1, pm_{i2}), \ldots, (sem_i, pm_{ij}), \ldots, (sem_m, pm_{in})\}, 1 < i < m, 1 < j < n \]

\[ C_{pm} \in \{\text{(SHRINK, ADD)}, \text{(SHRINK, REPLACE)}, \text{(ENLARGE, ADD)}, \text{(ENLARGE, REPLACE)}, \text{(MOVE, ADD)}, \text{(MOVE, DELETE)}, \text{(MOVE, REPLACE)}, \text{ZERO}\} \]

\[ O_{\text{sem.pm}} \in \{\text{ERASE&UPDIRS, UPDATE&UPDIRS, LOOKUP&UPDIRS, LOOKUP&DEPLOY&UPDIRS}\} \]

So the above expression of the evolution rule of entity services means that, given the set of relationship between sense-execute model and physical model, and a changed physical model and changing type, we can determine the operation for changing the sense-execute models and the relationship, and the relationship between sense-execute model and physical model after changing. The evolution procedure is depicted in Fig. 12. We express the
four procedures (i.e. ERASE&UPDIRS, UPDATE&UPDIRS, LOOKUP&UPDIRS, LOOKUP&DEPLOY&UPDIRS) by the processes of CSP. Each operation in the procedures, which is itemized in Fig. 12, is defined by a process separately, as described below.

(1) Process IRS to express \( R_{sem-pm} \) in the evolution mechanism, which includes only one event (i.e. “generate”).

(2) Process JUDGE to express the procedure of judging which operation is to be done. It includes two events (i.e. change and judge). The event “change” represents that the physical model has changed. Event “judge” is to judge the relationship between the changed physical model and the corresponding sense-execute model.

(3) Process ERASE corresponds to the ERASE operation, which includes two events (i.e. “erase” and “unlinkera”). The event “erase” represents that the items of \( \langle par, pps \rangle \) are deleted from the related sense-execute model. The event “unlinkera” represents removing the relationship between the sense-execute model and the physical model.

(4) Process UPDATE corresponds to the UPDATE operation, which includes only one event (i.e. “update”). The event “update” represents that the related items of \( \langle par, pps \rangle \) in sense-execute model are updated according to the changed physical model.

(5) Process LOOKUP corresponds to the LOOKUP operation, which includes three events (i.e. “unlink”, “search”, and “link”). The event “unlink” represents that the sense-execute model unlinks with the physical model. The event “search” represents that the changed physical model searches the related sense-execute model. The event “link” represents that the changed physical model is linked with the found sense-execute model.

(6) Process DEPLOY corresponds to the DEPLOY operation, which includes two events (i.e. “register” and “link”). The event “register” represents that the items of \( \langle par, pps \rangle \) of the physical model are registered to the new sense-execute model. The event “link” represents that the changed physical model is linked with a new sense-execute model.

F. Formal properties of \( fMES \)

Based on the above analysis of the processes and events in the procedures of the evolution mechanism, we can express them with CSP processes as follows.

(1) \( \text{IRS} \rightarrow \text{generate} \rightarrow \text{JUDGE}; \)

(2) \( \text{JUDGE}=\text{change} \rightarrow \text{judge} \rightarrow (\text{ERASE} \text{*} \text{UPDATE} \text{*} \text{LOOKUP}); \)

(3) \( \text{ERASE} \rightarrow \text{erase} \rightarrow \text{unlinkera} \rightarrow \text{ERAISA}; \)

(4) \( \text{UPDATE}=\text{update} \rightarrow \text{UPDISA}; \)

(5) \( \text{LOOKUP}=\text{unlink} \rightarrow \text{search} \rightarrow (\text{link} \rightarrow \text{LOOKISA} \text{*} \text{DEPLOY}); \)

(6) \( \text{DEPLOY}=\text{register} \rightarrow \text{link} \rightarrow \text{DPYISA}; \)

(7) \( \text{ERAISA}=\text{unlink} \rightarrow \text{IRS}; \)

(8) \( \text{UPDISA}=\text{pmudpirs} \rightarrow \text{IRS}; \)

(9) \( \text{LOOKISA}=\text{pmudpirs} \rightarrow \text{unlink} \rightarrow \text{link} \rightarrow \text{IRS}; \)

(10) \( \text{DPYISA}=\text{pmudpirs} \rightarrow \text{unlink} \rightarrow \text{link} \rightarrow \text{IRS}; \)

Then, we use PAT (Process Analytical Toolkit) [45] to verify the processes of \( fMES \), which are referred to as the evolution mechanism (EM) later. The results of verification are shown in Fig. 13. We can see that the process EM conforms to the properties of deadlock-free, divergence-free and nonterminating. Deadlock-free means that the evolution mechanism will not incur lock of process in the procedures of changing related sense-execute model and relationship set. Divergence-free means that it is impossible to incur state chaos for the evolution mechanism, that is to say, each process knows its next state in the evolution mechanism. Nonterminating means that the evolution process will not pause during its work. Therefore, it is verified that the method of maintaining entity service is correct in theory.

V. Case Study and Effectiveness Analysis

The IoT system taken as a case studied here consists of four environment monitoring entity services, as shown in Fig. 14, which are deployed in four areas (i.e. area A, area B, area C, and area D) of a city to provide physical data of the environment, namely \( \text{humidity} \), \( \text{temperature} \), \( \text{CO} \), \( \text{CO}_2 \) respectively. The four entity services are constructed according to PMDA and \( fDES \), so the environment monitoring parameters correspond to the physical model of PMDA.
which are named as pmA, pmB, pmC, and pmD respectively. Accordingly, the sense-execute models of the entity services are named as semA, semB, semC and semD, which provides process ability of humidity, temperature, CO, and CO2 respectively. Application models of these entity services are named as amA, amB, amC, and amD respectively.

![Diagram](image)

Fig. 14. The organization structure of the IoT system taken as a case study.

We take JCSP 1.1 [28] to develop and maintain an entity service based on \(fDES\) and \(fMES\). The reason why we take JCSP as the platform to do the case study is that software modules and linkers implemented with JCSP is close to real implementation, and it supports formal model checking based on CSP and \(\pi\)-calculus to synthetically verify effectiveness of developing and maintaining an entity service based on \(fDES\) and \(fMES\). The programming framework of JCSP is shown in Fig. 15, where we can see that each JCSP project is composed of components and linkers. A component consists of two parts. One part is to realize all the functions of the component, and the other part is interface for being connected with other related components. A link contains some channels for passing information from one component to another component. The channels can be used for establishing relationship among all the components.

**A. Implementation of entity services**

Due to space limited, we only take the “entity service A” shown in Fig. 14 as an example to be developed. Firstly according to \(fDES\) we implement the software modules in the entity service as components based on the programming framework of JCSP. Then, we implement channels in a linker for connecting these components. Finally, we connect all the components with the developed channels. In Fig. 16, we partially illustrate the implementation of the JUDGE-C software module in the environment monitoring entity service. We can see that the components JUD-P and LOC-P are connected by the channel named chan3.

![Diagram](image)

Fig. 15. The programming framework of JCSP.

![Diagram](image)

Fig. 16. Illustration of partial implementation of the JUDGE-C software module in the environment monitoring entity service.

**B. Implementation of maintenance method**

The main procedures of maintaining entity services according to \(fMES\) is also implemented in JCSP, as depicted in Fig. 17. In the center of the figure, the gray box contains the components and linkers for implementing \(fMES\), which corresponds to the ESMDM in the controller of software defined IoT shown in Fig. 1. More detailed description of ESMDM and its interaction with developed entity service is presented below.

(1) All the implemented entity services connect with ESMDM through channels.

(2) The component IRS in ESMDM corresponds to the process IRS in \(fMES\), which realizes the function of registering all the entity services to the SEM-PM relationship set.
(3) The component JUDGE in ESMDM corresponds to the process JUDGE in fMES, which realizes the function of detecting the physical parameters of all the entity services and judge the changing type of each entity service.
(4) The components ERASE, UPDATE and LOOKUP in ESMDM correspond to the process ERASE, UPDATE and LOOKUP in fMES respectively. If the component LOOKUP can find appropriate sense-execute model for the changed physical model, it will make association between the found sense-execute model and the changed physical model, and reconstruct the SEM-PM relationship set by calling the function of the component UPDIRS.
(5) If the component LOOKUP cannot find any sense-execute model for the changed physical model, the component DEPLOY will be called to deploy a new sense-execute model, and refresh the SEM-PM relationship set by calling the function of the component UPDIRS.
(6) The component UPDIRS updates SEM-PM relationship set for all the registered, added, or changed physical models of entity services.

C. Correctness verification

In the case study, we initially implement some entity services based on fDES and the main processes of fMES in JCSP, through linking 4 physical models (i.e. Temperature, Humidity, CO, and CO₂) with corresponding sense-execute models, which have the process ability (pro) of Temperature, Humidity, CO, and CO₂ respectively. The initial SEM-PM relationship set are kept in the ESMDM of the controller, as shown in Fig. 17. We also implement a sense-execute model (sem₁) to process physical parameters of CO and PM2.5 together. However, sem₁ is not initially registered into the SEM-PM relationship set in ESMDM. Then we test the results of fMES with some simulated scenarios by changing physical parameters of some specific entity services to trigger evolution of corresponding entity services. Taking the entity service with a physical parameter (pps) of CO as an example, the sequence of testing is listed in Table II.

<table>
<thead>
<tr>
<th>Changing pps (CO) to</th>
<th>Operation for changing the sense-execute models and the relationship</th>
<th>Tested Path of evolution mechanism (refer to Fig. 12)</th>
<th>SEM-PM relationship set in ESMDM</th>
</tr>
</thead>
<tbody>
<tr>
<td>CO+PM25</td>
<td>LOOKUP&amp;DEPLOY&amp;UPDIRS</td>
<td>(1),(2),(5),(6),(7)</td>
<td>Updated consistently with the changed pps</td>
</tr>
<tr>
<td>CO</td>
<td>UPDATE&amp;UPDIRS</td>
<td>(1),(4),(7)</td>
<td></td>
</tr>
<tr>
<td>CO₂</td>
<td>LOOKUP&amp;UPDIRS</td>
<td>(1),(2),(5),(7)</td>
<td></td>
</tr>
<tr>
<td>null</td>
<td>DEPLOY&amp;UPDIRS</td>
<td>(1),(2),(3),(4)</td>
<td></td>
</tr>
</tbody>
</table>

From Table II we can see that each group of testing sequence fully covers the possible cases of evolution mechanism. The results show that the SEM-PM relationship set can be updated in consistence with changed entity services, which verifies correctness of the proposed method of developing and maintaining entity services. Next, we will analyze how effective is our proposed method of maintaining entity service in software defined IoT.

D. Effectiveness analysis

In order to explain the effectiveness of developing entity services with fDES and maintaining them with fMES, we compare them with the way used to develop entity services with ASO [17] and reconstruct the ASO entity services when the required physical parameters change.

We assume that the cost for developing and maintaining an entity service with ASO is C₁. The number of entity services needed to construct an environment monitoring IoT system is N. Assuming that the expected times of changing requirements is M−1, the cost for developing and maintaining the system is C₈₀ = N * C₁ + (M−1) * N * C₁ = M * N * C₁.

According to fDES, sense-execute model and physical model need to be developed separately, while ASO does not have such a need. We assume that the extra cost for separate development of sense-execute model and physical model is α * C₁, where α is the ratio of the cost for separate developing sense-execute model and physical model with fDES to the cost for developing and maintaining an entity service with ASO. So the cost for developing and maintaining an entity service with fDES and fMES is (1 + α) * C₁. Two other costs for fDES and fMES are listed below.

1. The cost of developing and maintaining the ESMDM in the controller of software defined IoT is C₂.
2. The cost of registering all the entity services to ESMDM is N * C₃, where N is the number of entity services needed to construct an environment monitoring IoT system, C₃ is the cost for registering an entity service to ESMDM.
3. We use η to denote the ratio of the failure times to find appropriate sense-execute models to the times of requiring changes of physical parameters. Assuming that the expected times of changing requirements is M−1, the cost of deploying new sense-execute models is η * (M−1) * N * (1 + α) * C₁.

From the above analysis, we can see that the total cost of developing and maintaining entity services with fDES and fMES is C₈₀ = N * (1 + α) * C₁ + η * (M−1) * N * (1 + α) * C₁ + C₂ + N * C₃. We define the effectiveness of our
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2019.2929285, IEEE Internet of Things Journal

Lower cost than ASO for $N$ in cost. When $N$ is changed frequently, our proposed method would be better to use ASO. However, when the physical parameter $\eta$ is not changed so much, we reduce cost for developing and maintaining large-scale IoT models ($\eta$). We assume that $\alpha = 0.5$, $C_2/C_1 = 0.3$, $C_3/C_1 = 0.05$, based on experience of developing IoT system [51]. In particular, if the cost for developing and maintaining an entity service with ASO is 1 day*person, the cost for developing and maintaining an entity service with $fDES$ and $fMES$ is about 1.5 days*person. The averaged cost of developing and maintaining the ESMDM in the controller of software defined IoT for each entity service, and that of registering it to ESMDM can be approximately 0.3 day*person and 0.05 day*person, respectively.

Fig. 18 illustrates the ratio of $C_{PMDA}$ to $C_{ASO}$ ($\theta$) changes with the failure probability to find appropriate sense-execute models ($\eta$). $\theta$ changes with the number of entity services in system ($N$) is 3, if $\eta$ is 0.4, our proposed method has higher cost than ASO in the case that the expected times of changing requirements ($M$) is 2, but has lower cost than ASO in the case of $M = 20$. This means that if the IoT system is small and its physical parameter will not be changed so much, we had better to use ASO. However, when the physical parameter is changed frequently, our proposed method would be better in cost. When $M$ is 2 and $\eta$ is 0.4, our proposed method has lower cost than ASO for $N = 100$, but has higher cost than ASO for $N = 3$. This means that our proposed method can reduce cost for developing and maintaining large-scale IoT systems.

Fig. 19 illustrates the ratio of $C_{PMDA}$ to $C_{ASO}$ ($\theta$) changes with the times of changing requirements ($M$), when $\alpha = 0.5$, $C_2/C_1 = 0.3$, $C_3/C_1 = 0.05$, and $N = 100$. We can see that in all cases that $\eta$ is from 0.2 to 0.6, $\theta$ is not changed with increasing number of entity services. Hence, the effectiveness of our proposed method ($\theta$) is more sensitive to the times of changing requirements ($M$) than the number of entity services in system ($N$).

VI. CONCLUSION AND FUTURE WORKS

Entity service which abstracts sensing and executing devices in the physical space has been taken as a basic unit for constructing IoT systems. With more and more entity services occurring, a software-defined network approach has been applied in building IoT system, to ease managing and updating large-scale entity services. Currently, entity services are mostly developed with the same software architecture as traditional services on the Internet have an inherited problem in adaptability. In order to solve the problem, we considered common characteristics of entity services according to the ternary theory, and separated the functionalities of an entity service in social, cyber, and physical spaces from each other, and abstracted them into three models, which are application model, sense-execute model and physical model, and designed a Physical Model Driven software Architecture (PMDA). To ease development of entity services, we also proposed a Formal Development method of Entity Services ($fDES$) to transform the abstracted models of PMDA into implementable software modules. Besides, to reduce maintenance cost of
entity services when adapting them to different requirements from the social space and changing monitoring parameters of the physical space, we proposed a formal Maintenance method of Entity Service (fMES) to change, add or remove some inner software modules of entity services and link them together. Hence, fDES and fMES can be used for guiding implementation of entity services in the data plane and the entity service maintenance and deployment module (ESMDM) in the control plane of software-defined IoT respectively.

fDES was proved to satisfy three principles of loosely coupled design (i.e. Dependency Inversion, Law of Demeter, and Single Responsibility). The properties of maintaining entity services by fMES was checked by Process Analysis Toolkit (PAT), in terms of deadlock-free, divergence-free and non-terminating properties. The correctness of fDES and fMES was verified by a case study by implementing some entity services based on fDES and the main processes of fMES in JCSIP. We also proved that our proposed method was more effective in reducing the cost of developing and maintaining IoT applications systems, which are composed of large-scale entity services with frequent changes of requirements through analysis.

In the near future, we will implement some entity service based on fDES and the evolution mechanisms based on fMES in real testing platform, and compare their effectiveness with more related methods, such as developing entity services based on microservice architecture.

ACKNOWLEDGMENT

Partial work of this paper is supported by the Zhejiang Provincial Natural Science Foundation of China (LY18F020011), Ningbo Natural Science Foundation (2018A610154), the National Natural Science Foundation of China Provincial Natural Science Foundation of China (61672498) and the K.C. Wong Magna Fund in Ningbo University.

REFERENCES

Haiming Chen received the BEng and MEng degree in computer engineering from Tianjin University, China, in 2003 and 2006 and the PhD degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences in 2010. He is now an associate professor at the department of computer science, Ningbo University. From 2010 to 2017, he was a Research Assistant Professor with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing. He also was a visiting scholar from October 2015 for one year at the department of computer science, College of William and Mary, Williamsburg, VA, USA. He has authored or co-authored over 50 papers in areas of wireless, ad hoc, sensor networks and networked embedded computing systems. He was a recipient of the Outstanding Faculty Award from the Chinese Academy of Sciences in 2011 and 2016, the Third Prize of the Beijing Municipal Science and Technology Award in 2013, and the Best Paper of Chinese Journal of Computers (2013-2017). He has served as TPC member for several international conferences, such as IEEE ICCCN 2019, APCC 2019, COMNETSAT 2019, PerCom 2018-2019 WiP, and reviewers for several international journals, such as ACM Transactions on Sensor Networks, IEEE Internet of Things Journal, IEEE ACCESS, Elsevier Computer Networks, Ad Hoc Networks, and etc. He is a member of IEEE, ACM and CCF (China Computer Federation).

Kaibin Xie received the BS degree and MS degree in computer science from Northwestern Polytechnical University, China, in 2006 and 2009 respectively. He received the PhD degree in computer engineering from the Institute of Computing Technology, Chinese Academy of Sciences in 2016. He is now a lecturer at the North Minzu University, China. His research interests include software architecture of Internet of Things and ubiquitous computing.

Li Cui received her undergraduate degree from Tsinghua University, China, in 1985 and the MSc degree from the Institute of Semiconductors, Chinese Academy of Sciences, in 1988. She received the Ph.D. Degree from the University of Glasgow, UK, in 1999. She was a visiting scholar in the Department of Chemistry at the University of Toronto, Canada, in 1994. In 1995, she worked as a Royal Society Fellow and, from 1996 to 2003, a Research Associate in the Department of Electronics and Electrical Engineering at the University of Glasgow.

She received "The 100 Talents Program of CAS" award in 2004 from CAS and is currently a Professor and the leader of the wireless sensor network laboratory at the Institute of Computing Technology, Chinese Academy of Sciences. Her current research is focused on sensor technology, wireless sensor networks, and Internet of Things. She is an IET fellow and a member of the IEEE.

Antonio Pescapé is a Full Professor at the Department of Electrical Engineering and Information Technology, University of Napoli Federico II (Italy). He received the M.S. Laurea Degree in Computer Engineering and the Ph.D. in Computer and Control Engineering, both at University of Napoli Federico II. His research interests are in the networking field with focus on Internet Monitoring, Measurements and Management and on Network Security. He has co-authored over 200 journal (Communications of the ACM, JSAC, IEEE Network, IEEE Communications Magazine, IEEE Wireless Communications Magazine, etc.) and conference (SIGCOMM, IMC, PAM, Globecom, ICC, etc.) publications and he is co-author of several patents pending. He has served as workshops and conferences Chair and on more than 180 technical program committees of IEEE and ACM conferences. He has served as Editorial Board Member of IEEE Communications Surveys and Tutorials (2008-2011) and was guest editor for the special issue of Computer Networks on “Traffic classification and its applications to modern networks”. For his research activities he has received several awards. He is a Senior Member of the IEEE.