eMES: Easing Maintenance of Entity Services in Service Oriented Software-Defined Internet of Things

Haiming Chen†, Valerio Persico‡, Antonio Pescapè‡
†Faculty of Electrical Engineering and Computer Science, Ningbo University, Zhejiang, China 315211
‡Department of Electrical Engineering and Information Technology, University of Napoli Federico II, Napoli, Italy 80125
Email:chenhaiming@nbu.edu.cn, valerio.persico@unina.it, pescape@unina.it

***Abstract*—Based on the Service Oriented Architecture (SOA), Internet of Things (IoT) systems are usually developed by orchestrating application services and entity services, which are required to be easily adapted to meet varying requirements of sensing or controlling the physical space. To ease updating and modification of entity services, some software-defined network approaches have been applied in building IoT systems. However, in these software-defined IoT systems, entity services are usually developed with the same software architecture as traditional services in Internet, which are not so easy to be adapted. In order to solve the problem, we propose a Physical Model Driven software Architecture (PMDA) for guiding the design of entity services. Furthermore, to reduce the maintenance cost of the entity services when adapting them to different requirements generated from both the social and the physical space, we propose an *evolution Mechanism of Entity Services* (*eMES*). The correctness and effectiveness of *eMES* are verified by a case study and analysis respectively.**

***Index Terms*—Internet of Things, software defined, entity service, evolution mechanism.**

I. Introduction

With the rapid development of smart sensing, wireless networking, and embedded computing, more and more digital devices deployed in the physical world can be interconnected and interoperate to compose the cyber-physical-human Internet of Things (IoT) environment [1]. The goal of IoT is to connect any people and any things, in any places, at any time, in the form of services [2], which can be mainly categorized into two types, namely application services and entity services. The application services are those used for processing data, 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 [3], while entity services can be built at both the cloud and the device side (a.k.a. edge services [4]). These services are seen as ubiquitous ingredients to construct IoT systems supporting a huge number of application scenarios. According to the ternary theory [5], entity services—differently than traditional application services in the Internet—peculiarly consists of parts in three spaces, namely social, cyber, and physical space. In particular, entity services in the cyber space interact with people in the social space, who send requirements to devices in the physical space. Requirements from people usually consist in sensing data from the physical space and generating control information to change the status of devices in the physical space [6].

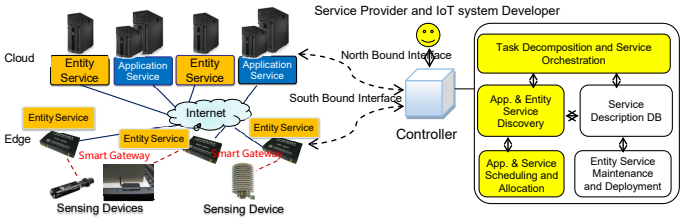

Fig. 1. Service Oriented IoT based on Software-Defined Network Architecture.

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 [7], [8]. 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 [9], [10]. However, the data plane of the service-oriented software-defined IoT [11] has hardly been changed from distributed objects (e.g. Physicalnet [12]), mobile agents (e.g. ASO [13]), or web services (e.g. SODA [14] and TinyREST [15]). Because they all encapsulate device- and scenario-specific functionalities into logic components in the same manner as business process modules, it is difficult to adapt them to the dynamic requirements from the social space and changing parameters of sensing or controlling the physical space.

To address the problem, some approaches have been proposed, such as supplying adaptation rules [16], [17] or self-adaptive methods [18] for the existing software architecture. These approaches need rules, models or solutions to meet dynamic requirements. However, it is hard to propose common rules, models or solutions for interacting with the social space and the physical space simultaneously. Therefore, it still

implies inefficiency of these approaches to interoperate with the control plane, especially the entity service maintenance and deployment module (ESMDM) shown in Fig. 1, to update entity services.

In this paper, based on our previously designed software architecture for developing IoT application systems [19], [20], [21], *(i)* we consider common characteristics of entity services according to the ternary theory, and abstract the functionalities of an entity service in social, cyber, and physical spaces into application model, sense-execute model and physical model, respectively, thus devising a Physical Model Driven software Architecture (PMDA) for guiding design of entity services; *(ii)* we propose an *evolution Mechanism of Entity Services* (*eMES*) to change, add or remove some inner software modules of entity services and connect them together, in order to reduce maintenance cost of entity services when adapting them to dynamic requirements from the social space and to the change of monitoring parameters of the physical space; finally, *(iii)* the correctness of *eMES* is verified by a case study and its effectiveness in reducing cost of developing and maintaining IoT application systems comprised of large-scale or frequently-changed entity services is verified by analysis.

The remainder of this paper is organized as follows. Section II describes the related work. Section III gives brief introduction of PMDA. Detail on *eMES* is presented in Section IV. Section V shows the results of correctness verification and effectiveness analysis of *eMES*. In Section VI, we make a conclusion.

II. Related Work

There are mainly three existing approaches that can be applied in maintaining entity services developed with distributed objects [12], mobile agents [13], or web services [14], [15], namely self-adaptive architecture [22], agent-based method [23] and modular update approach [24].

A *self-adaptive architecture* is built on a reusable framework which includes adaptive rules for the dynamic requirements. If the requirements of these software systems changes, the framework is able to monitor the changes and performs adaptation rules on these software systems.

Agent-based methods are essentially based on an interaction model to drive the agents to be adapted to the dynamic requirements. The interaction model includes interaction specification and interaction coordination, which are dynamically defined and amended according to the dynamic requirements of these software systems. Agent is illustrated by BDI (Belief, Desire, and Intention) model, social roles or knowledge and action, etc.

Finally, the *modular update approach* requires IoT software repositories to push meta-data information about software updates to IoT devices, which then select updates from a repository, and verify and install the received updates. It is proposed to partially update ROM of IoT devices, and be mainly used in IoT operating systems based on micro-kernel architecture.

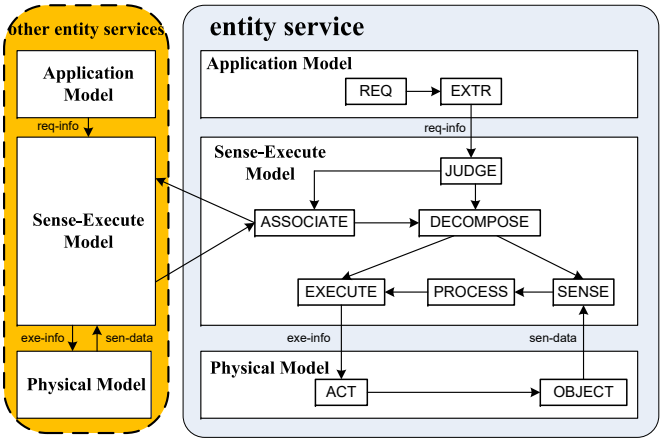

Fig. 2. Refined structure of PMDA.

These solutions can perform well in adapting software modules to dynamic requirements from the social space. However, they are still inefficient to interact with the social space and the physical space simultaneously, especially inefficient to interoperate with the control plane, the entity service maintenance and deployment module (ESMDM), to update entity services. In this paper, we 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.

III. 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, separating their functionalities in social, cyber, and physical spaces, and abstract them into three models: *application model*, *sense-execute model*, and *physical model*. 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. Therefore entity services in IoT application systems can parse the requirements of users, and take proper actions 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. 2.

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 is made of 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 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 consists of two components, 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.

IV. 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, followed with formal description of the *evolution Mechanism of Entity Service* (*eMES*).

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. 3, where *pmn* 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.

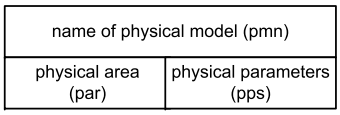

Fig. 3. Representation of a physical model.

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 [25], [26]. 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. $x = D$).

The three types of changing *par* in physical model are depicted by an example shown in Fig. 4.

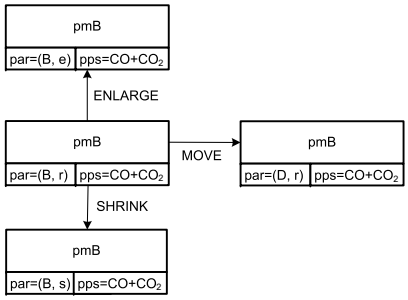

Fig. 4. An illustration depicting the three types of changing physical area (*par*) of physical model.

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 CO_2. 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 CO_2 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. CO_2), and adds some new sensors (i.e. pH).

The three types of changing *pps* in physical model are illustrated by an example shown in Fig. 5.

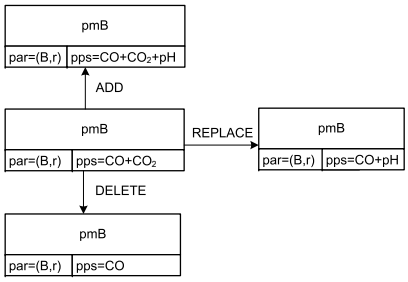

Fig. 5. An illustration depicting three types of changing physical parameters (*pps*) of physical model.

3) Changing both physical area and physical parameter: Based on the three changing types of *par* and three changing

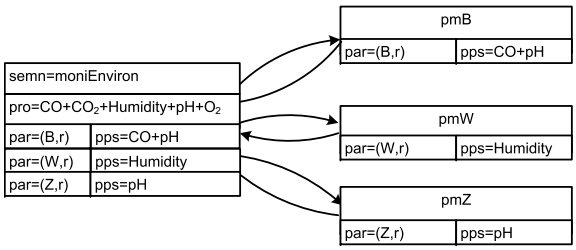

Fig. 6. Illustration of the relationship between a sense-execute model and physical models.

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, REPLACE), (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. 2, a sense-execute model interacts with a physical model to process physical parameters of an entity service. The relationship between a sense-execute model and physical models is illustrated in Fig. 6. The name of the sense-execute model is denoted as $semn$. The process ability of a sense-execute model is denoted as pro, which consists of several physical parameters it can process. We can see that more than one physical model 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 the 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. eMES *and its formal description*

Based on the above analysis of types of changing physical models, and corresponding operations for changing the sense-execute models, we design the evolution mechanism of entity service (*eMES*) 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 do not exist, deploy them.

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

$$\{R_{sem-pm}, (C_{pm}, pm'_j)\} \rightarrow \{O_{sem-pm}, R_{sem-pm}\},$$

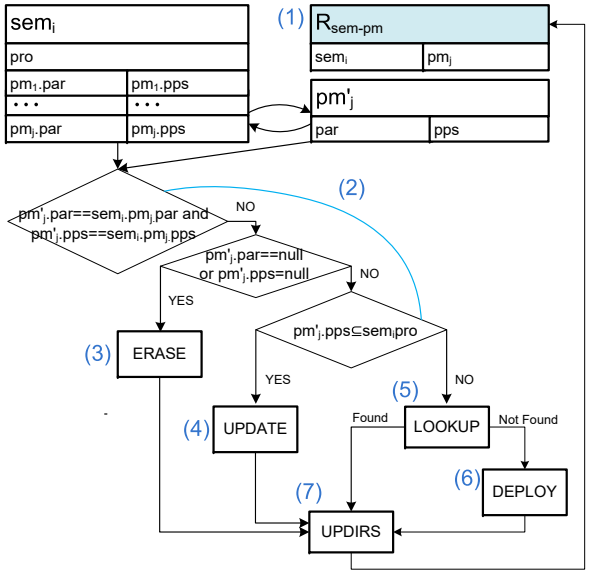

Fig. 7. Illustration of the procedure of the evolution mechanism of entity service.

where R_{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_{sem-pm} is the operation for changing the sense-execute models and the relationship.

R_{sem-pm}, C_{pm}, O_{sem-pm} are defined as follows.

$$R_{sem-pm} = \{(sem_1, pm_1), (sem_1, pm_2), \ldots, (sem_i, pm_j), \ldots, (sem_m, pm_n)\}, 1 < i < m, 1 < j < n$$

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

$$O_{sem-pm} \in \{\text{ERASE\&UPDIRS}, \text{UPDATE\&UPDIRS}, \text{LOOKUP\&UPDIRS}, \text{LOOKUP\&DEPLOY\&UPDIRS}\}$$

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.7. Each operation in the procedure 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.

(3) Process ERASE corresponds to the ERASE operation, which includes two events (i.e. 'erase' and 'unlinkera').

(4) Process UPDATE corresponds to the UPDATE operation, which includes only one event (i.e. 'update').

(5) Process LOOKUP corresponds to the LOOKUP operation, which includes three events (i.e. 'unlinkup', 'search', and 'link').

(6) Process DEPLOY corresponds to the DEPLOY operation, which includes two events (i.e. 'register' and 'link').

(7) Process ERAISA, UPDISA, LOOKISA, and DPYISA correspond to the UPDIRS operation after the ERASE, UPDATE, LOOKUP and DEPLOY operation respectively.

V. Correctness Verification and Effectiveness Analysis

The IoT system taken as a case study here consists of four environment monitoring entity services, as shown in Fig. 8, 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 $humidity, temperature, CO$, and CO_2 respectively. We take JCSP 1.1 [27] to develop and maintain an entity service based on PMDA and *eMES*.

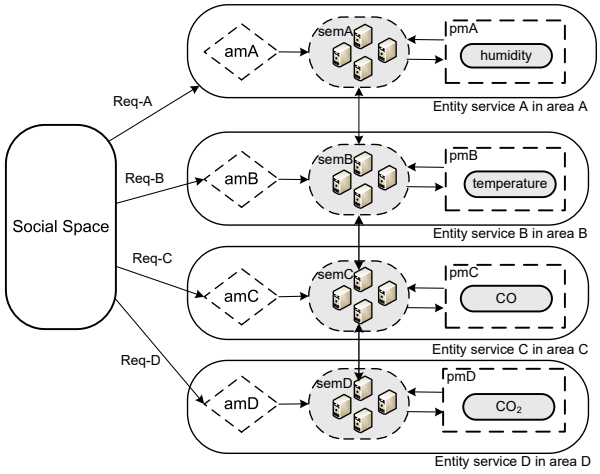

Fig. 8. The organization structure of the IoT system taken as a case study.

A. Implementation of entity services

Due to space limitations, Fig. 8 only shows the *entity service A* as an example to be developed. First we implement the software modules in the entity service according to PMDA as components based on the JCSP programming framework. Then, we implement channels in a linker for connecting these components. Finally, we connect all the components with the developed channels. In Fig. 9, 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.

Each entity service (ES) is developed through the approach presented above. In Fig. 10, we illustrate the implemented entity services with the main modules of the sense-execute model


```
public final class JUD-P implements CSProcess {
    boolean s=false;
    String App_type;
    private final ChannelInput in;
    private final ChannelOutput out;
    public JUD-P (ChannelInput in, ChannelOutput out){
        this.in=in;
        this.out=out;
    }
    public void run ()
    {
```

```
public final class LOC-P implements CSProcess {
    String req_content;
    private final ChannelInput in;
    private final ChannelOutput out;
    public LOC-P (ChannelInput in, ChannelOutput out){
        this.in=in;
        this.out=out;
    }
    public void run ()
    {
```

```
new JUD-P (chan2.in(), chan3.out());
new LOC-P (chan3.in(), chan5.out());
```

Fig. 9. Illustration of partial implementation of the JUDGE-C software module in the environment monitoring entity service.

(i.e. JUDGE-C, ASSOCIATE-C, DECOMPOSE-C, SENSE-C) and the main module of the physical model (i.e. OBJECT-C).

B. Implementation of maintenance method

The main procedures of maintaining entity services according to *eMES* is also implemented in JCSP, as depicted in Fig. 10. In the center of the figure, the gray box contains the components and linkers for implementing *eMES*, which corresponds to the "Entity Service Maintenance and Deployment Module" (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 *eMES*, 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 *eMES*, 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 *eMES* 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.

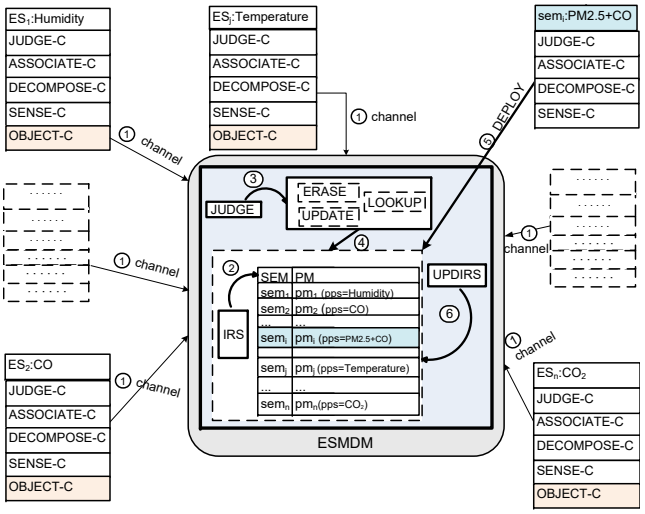

Fig. 10. Depiction of implemented maintenance procedures of *eMES* in JCSP 1.1.

(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 PMDA and the main processes of *eMES* in JCSP, through linking 4 physical models (i.e. $Temperature$, $Humidity$, CO, and CO_2) with corresponding sense-execute models, which have the process ability (*pro*) of $Temperature$, $Humidity$, CO, and CO_2 respectively. The initial SEM-PM relationship set are kept in the "Entity Service Maintenance and Deployment Module" (ESMDM) of the controller, as shown in Fig. 10. We also implement a sense-execute model (sem_i) to process physical parameters of CO and $PM2.5$ together. However, sem_i is not initially registered into the the SEM-PM relationship set in ESMDM. Then we test the results of *eMES* 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 I.

From Table I 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.

TABLE I
A GROUP OF TESTING SEQUENCE

Changing *pps* (CO) to	Operation for changing the sense-execute models and the relationship	Tested Path of evolution mechanism (refer to Fig. 7)	SEM-PM relationship set in ESMDM
CO+$PM2.5$	LOOKUP& DEPLOY&UPDIRS	(1),(2),(5),(6),(7)	Updated consistently with the changed *pps*
CO	UPDATE&UPDIRS	(1),(4),(7)	
CO_2	LOOKUP&UPDIRS	(1),(2),(5),(7)	
null	ERASE&UPDIRS	(1),(2),(3),(7)	

D. Effectiveness analysis

In order to explain the effectiveness of maintaining entity services with *eMES*, we compare them with the way used to develop entity services with ASO [13] 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 C_1. 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 $C_{ASO} = N*C_1+(M-1)*N*C_1 = M*N*C_1$.

According to PDMA, sense-execute model and physical model need to be developed separately, while ASO does not has such a need. We assume that the extra cost for separate development $\alpha*C_1$, where α is the ratio of the cost for separate developing sense-execute model and physical model with PMDA to the cost for developing and maintaining an entity service with ASO. So the cost for developing and maintaining an entity service with PMDA and *eMES* is $(1+\alpha)*C_1$. Two other costs for PMDA and *eMES* are listed below.

(1) The cost of developing and maintaining the "Entity Service Maintenance and Deployment Module" (ESMDM) in the controller of software defined IoT is C_2.

(2) The cost of registering all the entity services to ESMDM is $N*C_3$, where N is the number of entity services needed to construct an environment monitoring IoT system, C_3 is the cost for registering an entity service to ESMDM.

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 $\eta*(M-1)*N*(1+\alpha)*C_1$.

From the above analysis, we can see that the total cost of developing and maintaining entity services with PMDA and *eMES* is $C_{PMDA} = N*(1+\alpha)*C_1+\eta*(M-1)*N*(1+\alpha)*C_1+C_2+N*C_3$. We define the effectiveness of our proposed method as the ratio of C_{PMDA} to C_{ASO}, which is denoted by θ.

$$\theta = \frac{(1+\alpha)\cdot N\cdot C_1\cdot[\eta\cdot(M-1)+1]+C_2+N\cdot C_3}{M\cdot N\cdot C_1} = \frac{(1+\alpha)\cdot(\eta M-\eta+1)}{M}+\frac{1}{MN}\cdot\frac{C_2}{C_1}+\frac{1}{M}\cdot\frac{C_3}{C_1}$$

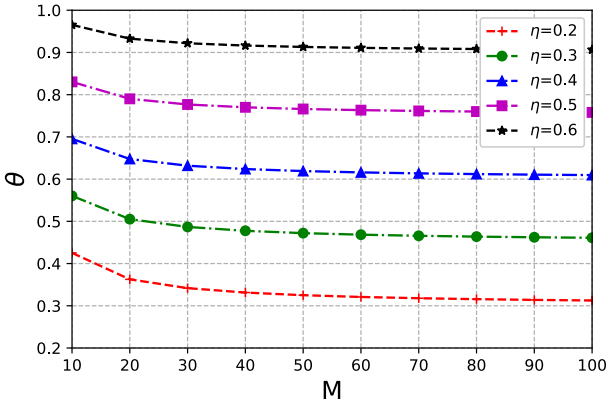

Fig. 11. The ratio of C_{PMDA} to C_{ASO} (θ) changes with the times of changing requirements (M).

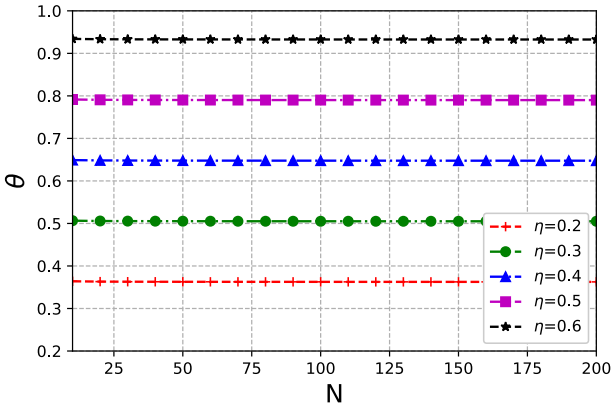

Fig. 12. The ratio of C_{PMDA} to C_{ASO} (θ) does not change with the number of entity services (N).

We assume that $\alpha = 0.5$, $\frac{C_2}{C_1} = 0.3$, $\frac{C_3}{C_1} = 0.05$, based on experience of developing IoT system. 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 PMDA and *eMES* 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. 11 illustrates the ratio of C_{PMDA} to C_{ASO} (θ) changes with the times of changing requirements (M). If $\theta < 1$, it means that our proposed method has lower development and maintenance cost than ASO. We can see that in all cases that η is from 0.2 to 0.6, θ is less than 1, which means that our proposed method is more effective than ASO in reducing development and maintenance cost. With increasing times of changing requirements, the effectiveness will be higher. Fig. 12 illustrates θ changes with the number of entity services (N). We can see that in all cases that η is from 0.2 to 0.6, θ is not changed with increasing number of entity services. Hence, the effectiveness of our proposed method (θ) is more sensitive to the times of changing requirements (M) than the number of entity services in system (N).

VI. Conclusion

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 in Internet have an inherited problem in adaptability. In order to solve the problem, we consider common characteristics of entity services according to the ternary theory, and separate the functionalities of a 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, and design a Physical Model Driven software Architecture (PMDA). Furthermore, to reduce maintenance cost of entity services when adapting them to different requirements from the social space and change monitoring parameters of the physical space, we propose an *evolution Mechanism of Entity Services* (*eMES*) to change, add or remove some inner software modules of entity services and link them together. We have implemented some entity services based on PMDA and the main processes of *eMES* in JCSP. Through a case study, we show the correctness of maintaining entity services by *eMES*. Besides, some analytical results show the effectiveness of *eMES* in reducing the cost of developing and maintaining IoT application systems, which are composed of large-scale entity services or with frequent changes of requirements.

Acknowledgment

Partial work of this paper is supported by the Zhejiang Provincial Natural Science Foundation of China (LY18F020011) and Ningbo Natural Science Foundation (2018A610154).

References

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of Things (IoT): A Vision, Architecture Elements, and Future Directions. *Future Generation Computer System*, 29(7):1645–1660, 2013.

[2] P. Banerjee, R. Friedrich, C. Bash, P. Goldsack, and et al. Everything as a Service: Powering the New Information Economy. *IEEE Computer*, 44(3):36–43, 2011.

[3] A. Botta, W. de Donato, V. Persico, and A. Pescapè. Integration of Cloud Computing and Internet of Things: A survey. *Future Generation Computer System*, 56(3):684–700, 2016.

[4] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637C646, 2016.

[5] D. Easley and J. Kleinberg. *Networks, Crowds, and Markets*. New York: Cambridge University Press, 2010.

[6] H. Zhuge. Semantic Linking through Spaces for Cyber-Physical-Social Intelligence: A methodology. *Artificial Intelligence*, 175(5):988–1019, 2011.

[7] Á. L. Valdivieso Caraguay, A. Benito Peral, L. I. Barona López, and et al. SDN: Evolution and Opportunities in the Development IoT Applications. *International Journal of Distributed Sensor Networks*, 2014.

[8] Z. Qin, G. Denker, C. Giannelli, P. Bellavista, and N. Venkatasubramanian. A Software Defined Networking architecture for the Internet-of-Things. In *Proc. IEEE NOMS*, pages 1–9, Krakow, 2014.

[9] Y. Jararweh, M. Al-Ayyoub, A. Darabseh, E. Benkhelifa, M. Vouk, and A. Rindos. SDIoT: A Software Defined based Internet of Things Framework. *Journal of Ambient Intelligence and Humanized Computing*, 6:453–461, 2015.

[10] Y. Li, X. Su, J. Riekki, T. Kanter, and R. Rahmani. A SDN-based Architecture for Horizontal Internet of Things Services. In *Proc. IEEE ICC*, pages 1–7, Kuala Lumpur, 2016.

[11] S. Tomovic, K. YoshigoeIvo, and M. Radusinovic. Software-Defined Fog Network Architecture for IoT. *Wireless Personal Communications*, 92(1):181–196, 2017.

[12] P. A. Vicaire, Z. Xie, E. Hoque, and J. A. Stankovic. Physicalnet: A Middleware for Programming Concurrent, across Administrative Domain Sensor and Actuator Networks. In *Proc. ACM SenSys*, pages 317–318, Berkeley, CA, USA, 2009.

[13] G. Fortino, G. Guerrieri, and W. Russo. Agent-oriented Smart Objects Development. In *Proc. IEEE CSCWD*, pages 907–912, Wuhan, China, 2012.

[14] S. de Deugd, R. Carroll, K.E. Kelly, B. Millett, and J. Ricker. SODA: Service-Oriented Device Architecture. *IEEE Pervasive Computing*, 5(3):94–96, 2006.

[15] T. Luckenbach, P. Gober, S. Arbanowski, A. Kotsopoulos, and K. Kim. TinyREST - A Protocol for Integrating Sensor Networks into the Internet. In *Proc. of the Workshop on Real-World Wireless Sensor Network (REALWSN)*, pages 1–7, Stockholm, Sweden, 2005.

[16] Y. Brun, G. D. MSerugendo, C. Gacek, and et al. Engineering Self-adaptive Systems Through Feedback Loops. In *Software Engineering for Self-adaptive Systems*, pages 48–70. Springer Berlin Heidelberg, 2009.

[17] M I. Campbell, J. Cagan, and K. Kotovsky. A Design: Theory and Implementation of An Adaptive, Agent-based Method of Conceptual Design. In *Artificial Intelligence in Design*, pages 579–598. Springer Netherlands, 1998.

[18] R. Agrawal, A. Arning, R. Seiffert, and et al. Self-adaptive Method and System for Providing a User-preferred Ranking Order of Object Sets. U.S. Patent: 6,370,526, 2002.

[19] K. Xie, H. Chen, and L.Cui. PMDA: a Physical Model Driven Software Architecture for Internet of Things. *Journal of Computer Research and Development*, 50(6):1185–1197, 2013.

[20] K. Xie, H. Chen, X. Huang, and L. Cui. Low Cost IoT Software DevelopmentIngredient Transformation and Interconnection. In *Proc. IEEE ICPADS*, Melbourne, Australia, Dec. 2015.

[21] K. Xie, H. Chen, D. Li, and L. Cui. The Evolution Mechanism for Dynamic Physical Models in the Internet of Things. In *Proc. IEEE SEKE*, Pittsburgh, USA, Jul. 2015.

[22] P. Oreizy, M.M. Gorlick, R.N. Taylor, and et al. An Architecture-based Approach to Self-adaptive Software. *IEEE Intelligent Systems*, (3):54–62, 1999.

[23] R. Arun, P. Davy, and B. Yolande. Enabling Self-learning in Dynamic and Open IoT Environments. *Procedia Computer Science*, pages 207–214, 2014.

[24] F. J. A. Padilla, E. Baccelli, T. Eichinger, and K. Schleiser. The Future of IoT Software Must be Updated. In *Proc. IAB Workshop on Internet of Things Software Update (IoTSU)*, Dublin, Ireland, Jun. 2016.

[25] J. Hwang, Y. Gu, T. He, and Y. Kim. Realistic Sensing Area Modeling. In *Proc. IEEE INFOCOM*, pages 2421–2425, Anchorage, Alaska, USA, May 2007.

[26] H. Mostafaei, A. Montieri, V. Persico, and A. Pescapé. An efficient partial coverage algorithm for wireless sensor networks. In *IEEE ISCC*, pages 501–506, 2016.

[27] P. H. Welch and A. W. P. Bakkers. Formal Analysis of Concurrent Java Systems. In *Proceedings of Communicating Process Architectures*, 2000.