2015 IEEE 21st International Conference on Parallel and Distributed Systems

Low Cost IoT Software Development—Ingredient Transformation and Interconnection

Kaibin Xie[1,2] , Haiming Chen[1] , Xi Huang[1] and Li Cui[1]
[1]Institute of Computing Technology, Chinese Academy of Sciences, Beijing
[2]University of Chinese Academy of Sciences, Beijing
Email: xiekaibin@ict.ac.cn, chenhaiming@ict.ac.cn, huangxi@ict.ac.cn and lcui@ict.ac.cn

***Abstract*—Sensing/actuating ingredients are software running on smart devices which are widely deployed in physical spaces and integrated into our daily life. An Internet of Things (IoT) application can be composed of multiple sets of sensing/actuating ingredients and each ingredient can satisfy specific requirement of users by processing specific physical data collected from physical world or committing an action. However, users' requirements for functionalities of ingredients and overall goals of IoT applications are diverse and may change. The application diversity leads to a rise in complexity as well as the cost of developing and maintaining the system software of an IoT application. In this work, we aim to reduce the design complexity and development cost of IoT application software. To achieve this goal, we use the software architecture PMDA which was proposed in our previous work. Here, we present a component-based formal methodology, namely FMDA, to transform the components of PMDA into sets of software modules as sensing/actuating functionality required. At the same time, we propose a connection mechanism, namely FMCA, which can match and establish the interconnection between any two sensing/actuating ingredients as required. The behavior of FMDA and FMCA have been investigated. The evaluation results show that FMCA is more effective in reducing the cost if the requirements changing frequently or the failure probability of matching the required sensing/actuating ingredients is small.**

***Keywords*—sensing/actuating ingredients; IoT application; formal methodology; transform; connection mechanism**

I. Introduction

With rapid development of smart objects [1], more and more sensing/actuating ingredients embedded in physical world can be interconnected and interoperated, composing a new computing environment, namely Internet of Things (IoT). IoT aims to connect any people and any things, in any places, at any time, by any networks and any services [2]. To achieve the target, more and more sensing/actuating ingredients are occurring in every walk of life.

An IoT application usually consists of several sensing/actuating ingredients. Each sensing/actuating ingredient can satisfy specific requirements for sensing functionality of users by processing collected physical data of physical environment. However, current researches mainly focus on developing and maintaining a single sensing/actuating ingredient of an IoT application. The users' requirements for the sensing functionalities and overall goal of an IoT application may change with the dynamic adjustment or/and additional physical parameters to be combined into the system. Also the users' requirements for sensing/actuating ingredients are diverse even sourcing from the same smart devices. This leads to a rise in complexity and cost of developing and maintaining the system software comprising an IoT application as required.

To address the problem, we propose a formal methodology to develop sensing/actuating ingredients, named FMDA and a mechanism to connect all the sensing/actuating ingredients into an IoT application, named FMCA. FMDA is based on PMDA [3] which is our previously proposed software architecture for developing sensing/actuating ingredients. FMCA is based on FMDA which can match and establish connection relationship between any two sensing/actuating ingredients as required. The main contributions of this paper are embodied in addressing challenges in designing FMDA and FMCA respectively.

There are two challenges in designing FMDA. Firstly, it is hard to transform the components of PMDA into software modules. We use Communication Sequential Process (CSP) [4] to realize the transformation. Secondly, it is difficult to prove that a sensing/actuating ingredient is easy to maintain if its development methodology conforms to FMDA. We prove that FMDA satisfies three principles which are Dependency Inversion Principle (DIP) [5], Law of Demeter (LoD) [6] and Single Responsibility Principle (SRP) [5]. Based on the three principles, we conclude that a sensing/actuating ingredient is easy to maintain if it conforms to FMDA.

There are also two challenges in designing FMCA. Firstly, it is hard to deliver the changed requirements to an appropriate sensing/actuating ingredient which exists in IoT application. We develop a connector module to connect and manage all sensing/actuating ingredients and solve the problem of delivering requirements between any two sensing/actuating ingredients as required. Secondly, it is hard to analyze the pros and the weakness of FMCA. We compare the cost of FMCA with the method, named FMDAI, which develops sensing/actuating ingredients according to FMDA but all these sensing/actuating ingredients are independently without any interconnection. We regard the cost of FMDAI as a benchmark. From analysis we get that FMCA is much better if the average number of requirements changes for each sensing/actuating ingredient is large, or the failure probability of matching the required sensing/actuating ingredients is small.

The remainder of the paper is organized as follows. In section II, we present the related work on adapting software systems to dynamic requirements. In section III, we review our previously proposed software architecture PMDA. In section IV, we propose FMDA for developing sensing/actuating ingredients and analyze the advantages of FMDA. In section V,

1521-9097/15 $31.00 © 2015 IEEE
DOI 10.1109/ICPADS.2015.14

we design the connection mechanism FMCA and evaluate the pros and the weakness of FMCA. Finally, we make a concluding remark and give future research directions.

II. Related work

In this section, we briefly introduce the related work of current approaches on adapting software systems to dynamic requirements. There are mainly three approaches on adapting software systems to dynamic requirements, including self-adaptive architecture [7] [8], agent-based method [9] [10] [11] and middleware-based self-adaptive method [12] [13] [14].

The self-adaptive architecture is a reusable framework to fulfill the dynamic requirements of several software systems [7] [8]. The framework is based on the software architecture of these software systems and designs adaptive rules for the dynamic requirements. If the requirements of these software systems change, the framework can monitor the changes and apply adaption rules to these software systems [7].

The adaptive agent method includes an interaction model of these software systems to drive the agents to address the dynamic requirements [9] [10]. The interaction model includes interaction specification and interaction coordination which are dynamically defined and amended according to the dynamic requirements of these software systems [9]. Agent is illustrated by BDI (belief, desire and intention) model [15], social roles [16], knowledge and action [17], and so on.

The self-adaptive middleware method is a solution to adjust or dynamically configure the context of these software systems in order to meet the dynamic requirements [12]. To design a better solution, we usually need to consider characteristics and businesses of these software systems [12] [13] [14].

The disadvantage of current approaches is that we have to change software modules when requirements change. To reduce the design complexity and development cost of IoT application when requirements change dynamically, we propose a low cost IoT software development solution, including a component-based formal methodology FMDA and a connection mechanism FMCA.

FMDA is based on our previously proposed the software architecture PMDA for IoT applications, and uses a formal method based on CSP to process the dynamic requirements of sensing/actuating ingredients. FMDA can achieve the same goal as the self-adaptive architecture method.

FMCA is based on FMDA, which can establish the interconnection between any two sensing/actuating ingredients as required. It enhances FMDA to meet dynamic requirements adaptively without changing the software modules of the related sensing/actuating ingredients.

III. An overview of the software architecture PMDA

The software architecture PMDA is proposed for developing IoT applications, which is described by ADL Wright [18]. It considers three common characteristics of IoT applications according to ternary theory [19]. Firstly, it considers the requirement information (req-info) in the social space. Secondly, it considers the sensing and execution information which is generated in the virtual space. Thirdly, it considers the sensed objects and actuators in the physical space. Basing on the three common characteristics, PMDA consists of three models, namely Application Model, Sense-Execute Model and Physical Model [3]. The framework of PMDA is shown in Fig. 1. Application Model sends requested information (req-info) to Sense-Execute Model. Sense-Execute Model receives req-info from Application Model and receives sensor data (sen-data) from Physical Model, and generates execution information (exe-info) for Physical Model according to the content of req-info and sen-data. Physical Model provides sen-data to Sense-Execute Model and receives exe-info from Sense-Execute Model to satisfy the req-info of Application Model.

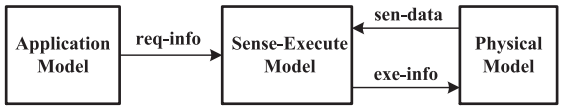

Fig. 1. The framework of PMDA

All the components of PMDA are illustrated in Fig. 2. Here we briefly introduce the components comprising the three models respectively. As shown in Fig. 2, Application Model has two components which are REQ and EXTR. REQ processes the requirements from the users and EXTR extracts the requirements which are related to the physical environment.

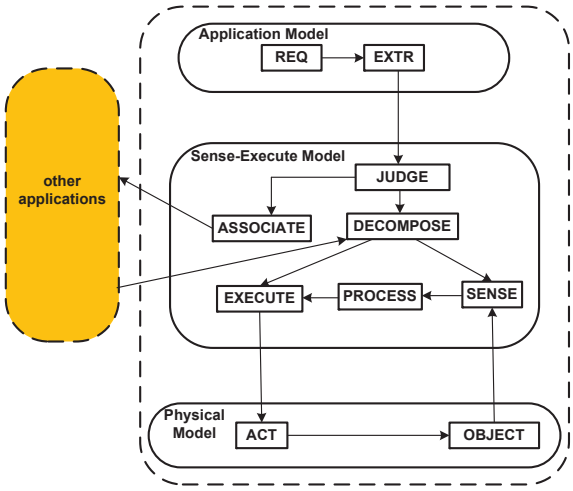

Fig. 2. Structure of PMDA

Sense-Execute Model has six components, including JUDGE, ASSOCIATE, DECOMPOSE, SENSE, PROCESS and EXECUTE. JUDGE divides the extracted requirements into two parts. One part is for the sensing/actuating ingredient itself and the other part is for the other sensing/actuating ingredients. ASSOCIATE processes the requirements which are related to other sensing/actuating ingredients. DECOMPOSE decomposes the requirements of the IoT application into two parts which are the requirements for sensing and the requirements for control. SENSE collects the related sensor data according to the requirements for sensing. PROCESS processes the sensor data and generates sensing information. EXECUTE generates the control information based on the requirements for control and the sensing information.

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

According to the structure of PMDA, we can see that JUDGE, ASSOCIATE and DECOMPOSE of PMDA are the key components to process the dynamic requirements of an IoT application.

IV. FMDA: FORMAL METHODOLOGY FOR DEVELOPING A SENSING/ACTUATING INGREDIENT

In this section, we design a formal methodology FMDA for developing a sensing/actuating ingredient based on the software architecture PMDA. It includes the following two steps: firstly, based on CSP which can establish models of complex interactions between concurrent software modules [20], we transform the components of PMDA into software modules. Secondly, we analyze the relationship of these software modules and connect them together to form a software system of a sensing/actuating ingredient. As the correctness of PMDA [3] has been verified, we can guarantee that the software system of a sensing/actuating ingredient is logically correct if the sensing/actuating ingredient conforms to FMDA.

Based on the above two steps, we prove that FMDA conforms to three principles which are Dependency Inversion Principle(DIP) [5], Law of Demeter(LoD) [6] and Single Responsibility Principle(SRP) [5]. According to the three principles, we conclude that a sensing/actuating ingredient is easy to maintain if it conforms to FMDA.

A. Transform components into software modules

In order to transform the abstract components of PMDA into concrete software modules, we analyze the elements of PMDA and transform them into corresponding software modules.

There are three elements in PMDA, including component, process and event. The meanings of them 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 IoT application.

The relationship of the three elements is as follows. Event is the basic element for process and component. A process is composed of several events and all the events in a process are executed in sequential. 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 sequential. The processes in different groups are executed in parallel.

The software modules for event, process and component are respectively named as Module-E, Module-P and Module-C respectively. The symbols of these modules and their interfaces are depicted in Fig. 3. The structure of Module-P and Module-C and their interfaces are depicted in Fig. 4. Module-E realizes the functions of an event, Module-P realizes the functions of a process, and a Module-C realizes the functions of a component. According to the structure shown in Fig. 4, a Module-P contains several Module-Es; a Module-C is composed of several Module-Ps. In order to connect Module-Es in 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.

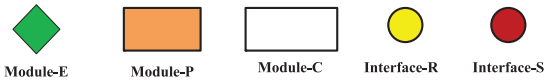

Fig. 3. Symbols of Module-E, Module-P, Module-C, Interface-R and Interface-S

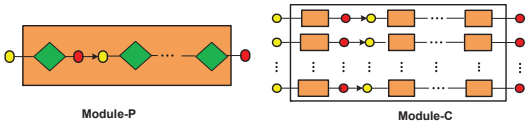

Fig. 4. The structures of Module-P, Module-C and their interfaces

B. The Relationships of Module-Es

In order to develop a sensing/actuating ingredient, we need to connect all Module-Es of the application by their interfaces (Interface-R or Interface-S). According to the structure of Module-P and Module-C, there are five situations for Module-Es to interact. First is that two Module-Es interact in the same Module-P. Second is that two Module-Es interact between two Module-Ps of the same Module-C. Third is that a Module-E of one Module-C interacts with a Module-E of another Module-C. Fourth is that a Module-E of one Module-C interacts with two or more Module-Es of another Module-C. Fifth is that two or more Module-Es of one Module-C interact with a Module-E of another Module-C.

Based on the five situations, we conclude that there are three relationships for Module-Es, including ONE-TO-ONE, ONE-TO-MANY and MANY-TO-ONE. We introduce the three relationships as follows.

ONE-TO-ONE relationship denotes that Interface-S of one Module-E outputs all its generated information only to Interface-R of another Module-E. The first, the second and the third situations belong to ONE-TO-ONE relationship. ONE-TO-ONE relationship is depicted in Fig. 5.

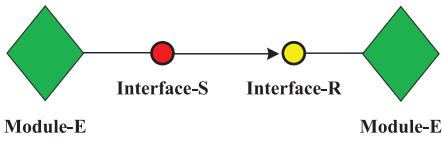

Fig. 5. ONE-TO-ONE relationship

ONE-TO-MANY relationship denotes that Interface-S of a Module-E outputs all its generated information to two or more Interface-Rs of Module-Es. The fourth situation belongs to ONE-TO-MANY relationship. ONE-TO-MANY relationship is depicted in Fig. 6.

MANY-TO-ONE relationship denotes that two or more Interfaces-Ss of Component-Es output all their generated information to Interface-R of a Module-E. The fifth situation belongs to MANY-TO-ONE relationship. MANY-TO-ONE relationship is depicted in Fig. 7.

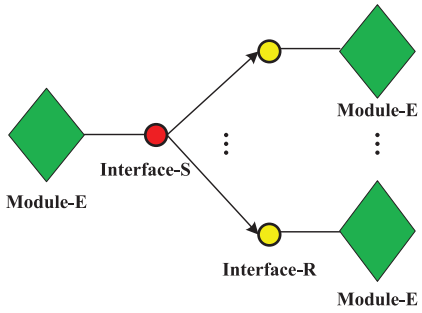

Fig. 6. ONE-TO-MANY relationship

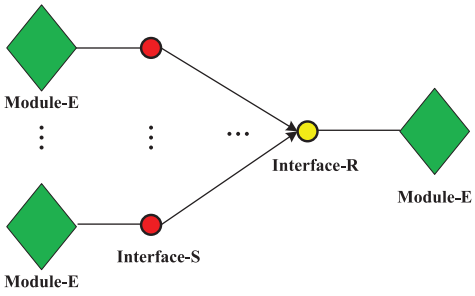

Fig. 7. MANY-TO-ONE relationship

C. The advantages of FMDA

In this section, we prove that FMDA satisfies three principles and conclude the advantages of FMDA.

Firstly, we prove that FMDA satisfies DIP. 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 [5]. According to FMDA, the high-level module can be regard as Module-E which receives information from other Module-Es. The low-level module can be regard as Module-E which sends its information to the related Module-Es. In a sensing/actuating ingredient, the high-level module interacts with the low-level module only through Interface-R and Interface-S. That is to say, the high-level module is independent of the low-level module and both dependent on the software architecture PMDA. So FMDA satisfies the principle of DIP.

Secondly, we prove that FMDA satisfies LoD. LoD explains that each module in a sensing/actuating ingredient 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 FMDA, each Module-E only interacts with other Module-Es when they must interact to finish the behavior of a task or 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 other Module-Es. So FMDA satisfies LoD.

Finally, we prove that FMDA satisfies SRP. SRP requires that each module of a sensing/actuating ingredient should have only one responsibility. According to FMDA, each Module-E represents only one action of a behavior. An action of a behavior can be regard as a responsibility of a sensing/actuating ingredient. So FMDA satisfies SRP.

According to DIP, the high-level Module-E is independent from the low-level Module-E in a sensing/actuating ingredient. According to LoD, all Module-Es are high-cohesion. According to SRP, all the Module-Es are low-coupling.

According to the above analysis, we can see that each Module-E in a sensing/actuating ingredient has three properties which are highly independent, high-cohesion and low-coupling. According to the three properties, we can conclude that a sensing/actuating ingredient can easily maintain if it conforms to FMDA.

D. Develop a single sensing/actuating ingredient according to FMDA

Taking an environmental monitoring application as example, we develop a single sensing/actuating ingredient according to FMDA. The goal of the sensing applications is to monitor the temperature and the humidity of a room and make people feel comfortable in the room. We consider that the temperature from 23 to 27 and the humidity from 40 to 60 are a comfortable conditions.

As shown in Fig. 1, there are ten components in PMDA, we need to develop ten Module-Cs according to FMDA. The ten Module-Cs in this application are REQ-C, EXTR-C, JUDGE-C, ASSOCIATE-C, DECOMPOSE-C, SENSE-C, PROCESS-C, EXECUTE-C, OBJECT-C and ACTUATE-C. Due to limited space, we only present development of Module-Es and Module-Ps in REQ-C through component transformation. For other Module-Cs, we only present their Module-Ps. The flow diagram of all Module-Cs and their Module-Ps is depicted in Fig. 8.

The task of REQ-C is to process the requirements of users in the room. REQ-C has only one Module-P, named REQ-P. Therefore all the functions of REQ-C are realized by REQ-P. REQ-P has three Module-Es, namely RECV-E, PROC-E and SEND-E. The action of RECV-E is to receive the requirements of the users. The action of PROC-E is to process the requirements of the users. The action of SEND-E is to send the processed requirements to module EXTR-C.

The task of EXTR-C is to get the extracted requirements according to the processed requirements. EXTR-C interacts with JUDGE-C which is in the virtual space. EXTR-C has only one Module-P, namely EXTR-P. Therefore all the functions of EXTR-C are realized by EXTR-P.

The task of JUDGE-C is to judge extracted requirements according to the temperature and humidity of the room and sends extracted requirements to ASSOCIATE-C or DECOMPOSE-C according to the content of extracted requirements. JUDGE-C has three Module-Ps named as JUD-P, LOC-P and EXT-P respectively. The behavior of JUD-P is to judge the extracted requirements according to the temperature and humidity of the room. If extracted requirements are related with the temperature and humidity of the room, JUD-P sends extracted requirements to LOC-P; otherwise, it sends extracted requirements to EXT-P. The behavior of LOC-P is to send requirements of the application to DECOMPOSE-C. The

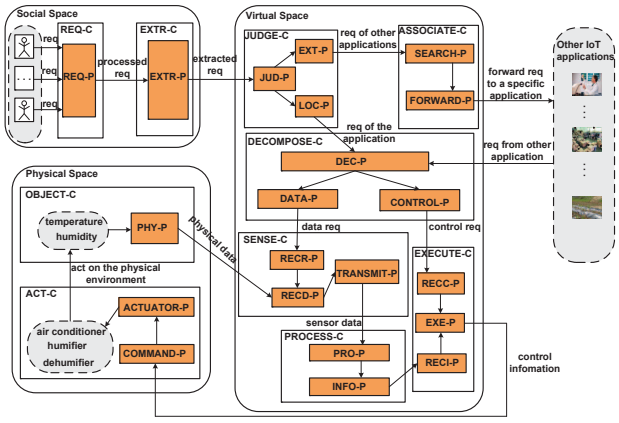

Fig. 8. The flow diagram for all Module-Cs and their Module-Ps

behavior of EXT-P is to send the requirements, which have nothing to do with the application itself, to ASSOCIATE-C.

The task of ASSOCIATE-C is to search the related IoT application according to the requirements for other IoT applications and forward these requirements to a related IoT application. ASSOCIATE-C has two Module-Ps, named as SEARCH-P and FORWARD-P respectively. The behavior of SEARCH-P is to search the related IoT application according to requirements for other sensing/actuating ingredients. The behavior of FORWARD-P is to forward requirements to the related IoT application.

The task of DECOMPOSE-C is to decompose all the requirements of the application into two parts, one part is related with the data requirements of temperature or humidity; the other part is related with the control requirements of the temperature or humidity. The requirements are from the users of the application and the other IoT applications. The requirements from other sensing/actuating ingredients are related to the humidity or temperature of the room. DECOMPOSE-C has three Module-Ps, named as DEC-P, DATA-P and CONTROL-P respectively. The behavior of DEC-P is to decompose the all the requirements of the application. The behavior of DATA-P is to receive the requirements which are related to the temperature data or humidity data of the room. The behavior of CONTROL-P is to receive the control requirements according to the requirements of the application.

The task of SENSE-C is to collect the sensor data (temperature and humidity) of the room according to the requirement data and send these sensor data to PROCESS-C. SENSE-C interacts with OBJECT-C which is in the physical space. It has three Module-Ps, named as RECR-P, RECD-P and TRANSMIT-P respectively. The behavior of RECR-P is to receive the requirement data. The behavior of RECD-P is to collect the sensor data according to the requirement data. The behavior of TRANSMIT-P is to transmit the sensor data to PROCESS-C.

The task of PROCESS-C is to process the sensor data and sends the generated sensor information to EXECUTE-C. PROCESS-C has two Module-Ps, named as PRO-P and INFO-P respectively. The behavior of PRO-P is to process the temperature data and humidity data, generating sensor information. The behavior of INFO-P is to send the sensor information to EXECUTE-C.

The task of EXECUTE-C is to generate the execution commands according to requirement control and the sensor information. EXECUTE-C interacts with ACT-C which is in the physical space. EXECUTE-C has three Module-Ps, named as RECC-P, RECI-P and EXE-P respectively. The behavior of RECC-P is to receive requirement control. The behavior of RECI-P is to receive the sensor information. The behavior of EXE-P is to generate the control commands according to the requirement control.

The task of OBJECT-C is to provide the sensor data which is the temperature and humidity value sensed in the room. OBJECT-C has only one Module-P named PHY-P. Therefore all the functions of OBJECT-C are realized by PHY-P.

The task of ACT-C is to receive control commands and trigger related actuators accordingly. There are three actuators in the room, including an air conditioner for controlling temperature, a humidifier and a dehumidifier for controlling humidity. ACT-C has two Module-Ps, named COMMAND-P and ACTUATOR-P respectively. The behavior of COMMAND-P is to receive the control commands from the controllers of EXE-P and send these commands to ACTUATOR-P. The behavior of ACTUATOR-P is to receive commands from COMMAND-P and control the related actuators accordingly.

As far we have got all Module-Es of the environmental monitoring application according to Module-Ps. We can connect all Module-Es according to the three relationships of FMDA and form a software system of the sensing/actuating ingredients.

E. Maintain the sensing/actuating ingredients

The requirements of users are likely to change due to the ever-changing environment. Once the requirements changed, we should modify the environmental monitoring application accordingly. We propose a solution to analyze the changing parameters of the sensing/actuating ingredients. We propose a solution to maintain the changed application based on FMDA.

For example, someday users also want to monitor PM2.5 of the room. So they would like to deploy air-quality monitor to sense PM2.5 value, and use air purifier to reduce air pollution when the value is higher than 80g/m^3.

To meet such requirement changes, we can extend the previous environment monitoring application by adding new Module-Cs and changing the interfaces relatively. A flow chart for the changed modules and the new added modules of this application is depicted in Fig. 9. This solution includes three procedures as following:

Firstly, the changed environmental monitoring application is directly related with the PM2.5 monitor and air purifier. We need to add a module named OBJECT2.5-C to monitor the values of PM2.5, and a module named ACT2.5-C to control the air purifier and reduce air pollution when PM2.5 is higher than 80g/m^3.

Secondly, SENSE-C, PROCESS-C, and EXECUTE-C are closely related with OBJECT2.5-C and ACT2.5-C. We need

to add new Module-Ps and change related Module-Ps of the three modules.

For SENSE-C, we add a new Module-P, namely DIVSEN-P, which divides the requirements of RECR-P into two parts. One part is to collect the data of temperature and humidity, which is the existing module RECD-P; the other part is to collect the data of PM2.5, namely RECD2.5-P. We also need to add a new Module-P, namely TRANSMIT2.5-P, which sends the PM2.5 data to PROCESS-C. RECR-P and RECD-P are the existing modules. We should change their interfaces to interact with the new module DIVSEN-P. For PROCESS-C, we need to add a new Module-P to process the PM2.5 data, namely PRO2.5-P. We also need a new Module-P to transmit the sensor information generated by PRO2.5-P, namely INFO2.5-P.

For EXECUTE-C, we need to divide the control requirements of the existing module RECC-P into two parts, one part is for the temperature and humidity data, namely CONTEM-P; the other part is for the PM2.5 data, named CON2.5-P. We also need two new Module-Ps, one named REC2.5-P is to receive PM2.5 from the air-quality monitor, the other one, namely EXE2.5-P, is to generate the control command for the air purifier accordingly. Therefore we should change the interfaces of RECC-P and EXE-P respectively.

Thirdly, JUDGE-C, ASSOCIATE-C, and DECOMPOSE-C are directly associated with the requirements. We need to add new Module-Ps and change the related Module-Ps according to the changed requirements. For JUDGE-C, We need to add a new Module-P named JUD2.5-P, which judges requirements from EXT-P. If requirements are related to PM2.5, JUD2.5-P sends these requirements to the existing module LOC-P. Otherwise the requirements are sent to ASSOCIATE-C. We use JUD2.5-P to interact with ASSOCIATE-C directly by deleting the interface of EXT-P to interact with ASSOCIATE-P. EXT-P has a new interface to send the related requirements. For ASSOCIATE-C, the existing module SEARCH-P should receive requirements from EXT2.5-P of JUDGE-C rather than form EXT-P of JUDGE-C. For DECOMPOSE-C, the existing module DEC-P has new interface to receive the requirements related to PM2.5 from the other IoT applications. REQ-C and EXTR-C in the social space have nothing to do with the changed sensing applications, so they do not need to change. Basing on the three procedures of the solution, we can change the environmental monitoring application, and adding the function for monitoring PM2.5 in the room.

V. FMCA: CONNECTION MECHANISM FOR IOT APPLICATION

In order to save the cost of adapting an IoT application to diverse requirements of different users, we propose a connection mechanism, named FMCA, which can establish interconnections between any two sensing/actuating ingredients as required.

We assume that the sensing/actuating ingredients of an IoT application are deployed in different physical locations. We develop a connector module, named LINKER, to record the name and the physical parameters of these sensing/actuating ingredients. LINKER has interfaces to interact with sensing/actuating ingredients of an IoT application.

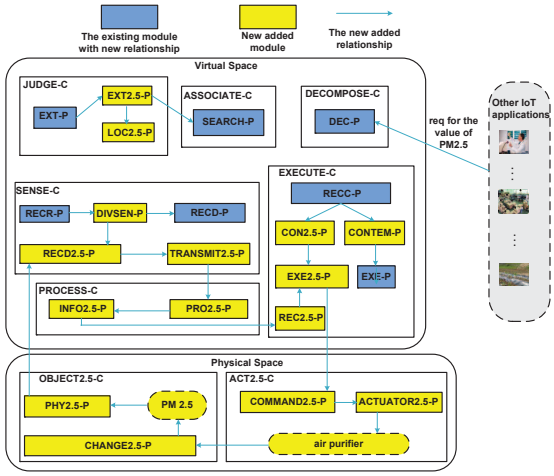

Fig. 9. The flow chart for the changed modules and the new added modules of the environmental monitoring application

A. The procedures of FMCA

The function of LINKER is to identify all the sensing/actuating ingredients, recording name and processing ability of them. There are three processes for FMCA, including REGISTRATION, INTERACTION and DEVELOP.

The process REGISTRATION is to register name and corresponding physical parameters of these sensing/actuating ingredients. The physical parameters denotes the processing ability of the sensing/actuating ingredients. The process REGISTRATION of all the sensing/actuating ingredients in an IoT application is depicted in Fig. 10.

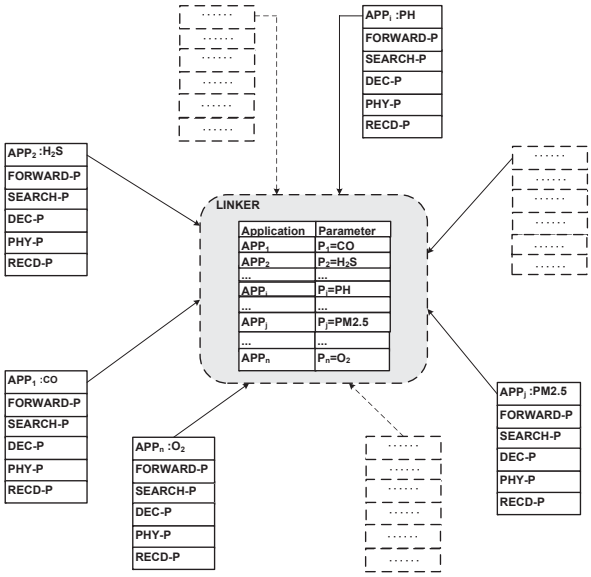

Fig. 10. The process REGISTRATION of all the sensing/actuating ingredients in an IoT application

The process INTERACTION denotes that we can find the matched sensing/actuating ingredients for the changed

requirements. There are five steps in INTERACTION. In the first step, APP_i delivers the changed requirement (Monitor the value of PM2.5) to LINKER. In the second step, LINKER returns DEC-P of APP_j to FORWARD-P of APP_i . LINKER returns RECD-P of APP_j to PHY-P of APP_i in the third step. The fourth step is to send the requirement of APP_i to DEC-P of APP_j. The last step is to send the requirement of the PM2.5 data of APP_i to RECD-P of APP_j. The five steps of process INTERACTION is depicted in Fig. 11 .

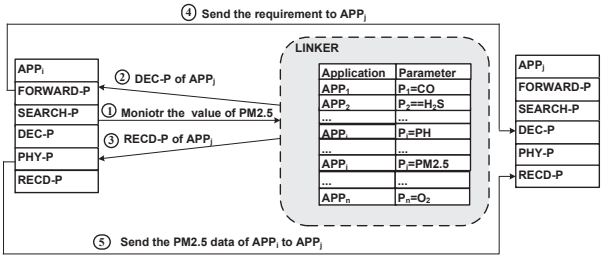

Fig. 11. The five steps of process INTERACTION

The process DEVELOP denotes that there is no matched sensing/actuating ingredients for a specific requirement. There are three steps in DEVELOP. The first step is the same as the first step in INTERACTION. The second step is to notify the developer to develop new software modules for the changed requirements of APP_i. In the last step, after the sensing/actuating ingredient has the ability to process the new physical parameter, we should register the new physical parameter for the sensing application. The three steps of process DEVELOP is depicted in Fig. 12.

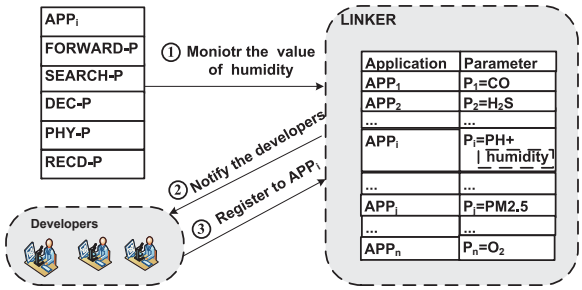

Fig. 12. The process DEVELOP for the unmatched requirements

B. The advantages of FMCA

In order to explain the advantages of FMCA in processing dynamic requirements, we propose another method for comparison, named FMDAI. It develops sensing/actuating ingredients according to FMDA but all these sensing/actuating ingredients are independently without any relationship.

According to the above description about FMDAI, there is no need to analyze the requirements of sensing/actuating ingredients even if the requirements have changed. That is to say, in order to process the changed requirements of a sensing/actuating ingredient, we only need to develop SENSE-C, PROCESS-C and EXECUTE-C in the virtual space. We assume that the cost for developing or maintaining a sensing/actuating ingredient is C_1 according to FMDAI, the number of the sensing/actuating ingredients is N and the number of the average requirements changes for each sensing/actuating ingredient is M-1. The cost for developing or maintaining an IoT application according to method FMDAI is P_{FMDAI}, and

$$P_{\text{FMDAI}} = N * C_1 + (M-1) * N * C_1 = M * N * C_1 \quad (1)$$

According to FMCA, we need to develop all Module-Cs in the virtual space. That is to say, except the Module-Cs in FMDAI, we also need to develop JUDGE-C, ASSOCIATE-C and DECOMPOSE-C which are related to the requirements while FMDAI has no need to develop them. We assume that the cost for developing or maintaining JUDGE-C, ASSOCIATE-C and DECOMPOSE-C of a sensing/actuating ingredient is α* C_1, where α is the ratio of the cost for developing or maintaining the three Module-Cs related to the requirements to C_1. So the cost for developing or maintaining a sensing/actuating ingredient is (1+α)*C_1 according to FMCA. We also need to develop the connector module LINKER for an IoT application. We assume that the cost for developing or maintaining LINKER is C_2. We need to establish the relationship of LINKER with sensing/actuating ingredients. We assume that the cost for developing or maintaining a sensing/actuating ingredient is C_3. Because the number of the sensing/actuating ingredients is N, the cost for the connection of all the sensing applications with LINKER is N*C_3. Assuming that η is the failure probability of forwarding requirements, which denotes the ratio of the number of the changed requirements that cannot be processed by any sensing/actuating ingredients to all the changed requirements. We use the same assumption from FMDAI, the cost for FMCA is P_{FMCA} and

$$\begin{aligned} P_{\text{FMCA}} = N * (1+\alpha) * C_1 + \eta * (M-1) * N \\ *(1+\alpha) * C_1 + C_2 + N * C_3 \end{aligned} \quad (2)$$

where θ denotes the ratio of P_{FMCA} to P_{FMDAI} and we can get that

$$\begin{aligned} \theta = (1+\alpha) * [(\eta * M + 1 - \eta)/M] + [1/(M * N)] \\ *(C_2/C_1) + (1/M) * (C_3/C_1) \end{aligned} \quad (3)$$

Based on our experience in developing IoT applications, we give a reasonable assumption that α=0.5, C_2/C_1=0.3, and C_3/C_1=0.05.

The worst case for FMCA is that M=1 and N=1, which denotes that there is only one sensing/actuating ingredient in an IoT application and the requirements of the IoT application remain unchanged. For the worst case, θ =(1+α)+C_2/C_1+C_3/C_1. Based on the above assumption, θ_{max}=1.85 and η is 1 in the worst case.

The best case for FMCA is that M is big enough, which denotes that the requirements have changed many times for each sensing/actuating ingredient. For the best case, θ =(1+α)*η. That is to say, θ can be very small if η is very small.

Fig. 13 illustrates η vs θ with different M and N. We can see that the smaller value for M or N, the bigger value for θ if other condition remains unchanged. We can further see that if the average number of requirements changes(M-1) for each sensing/actuating ingredient is large or the failure probability

of forwarding requirements(η) of an IoT application is small, FMCA is much better than the method FMDAI.

According to the Fig. 13, we can get the conclusion that we should use FMCA to reduce the cost of an IoT application if θ <1, otherwise, we should use FMDAI.

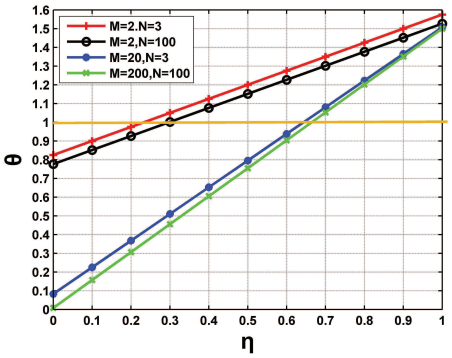

Fig. 13. θ (the ratio of P_{FMCA} to P_{FMDAI}) vs η (the failure probability of forwarding requirements), where the line θ=1 means that FMCA is equal to FMDAI. If θ<1, it means that FMCA is better than FMDAI. If θ>1, it means that FMCA is worse than FMDAI

VI. Conclusion and Future Works

An IoT application usually consists of several sensing/actuating ingredients. Each sensing/actuating ingredient can satisfy specific requirements for sensing functionality of the users by processing collected physical data of the physical environment. A single sensing/actuating ingredient could not always satisfy users' requirements. The application diversity leads to a rise in complexity as well as the cost of developing and maintaining the system software of an IoT application. In this work, we propose a component-based formal methodology, namely FMDA, to reduce the complexity and cost of developing or maintaining sensing/actuating ingredient. We have proved that a sensing/actuating ingredient is easy to maintain if it conforms to FMDA. Based on FMDA, we put forward a connection mechanism FMCA to connect any two sensing/actuating ingredients as required. FMCA is of advantage in reducing the development cost when the average number of requirement changes for each sensing components is large, or the failure probability of matching the required sensing/actuating ingredients is small. Methodology and mechanism provided in the paper apply to an IoT application which consists of many dynamic sensing/actuating ingredients and can process most dynamic requirements.

As for future works, we would consider how to adapt system software of IoT applications to the dynamic requirements with business relationship taken into consideration [21].

Acknowledgment

This paper is supported in parts by the National Science Foundation of China(NSFC) under Grant No.61100180 and No.61303246, the International S&T Cooperation Program of China (ISTCP) under Grant No.2013DFA10690.

References

[1] G. Kortuem, F. Kawsar, D. Fitton, et al. *Smart objects as building blocks for the internet of things*. Internet Computing, pp.44-51, 2010.

[2] O. Vermesan, P. Friess, P. Guillemin, et al. *Internet of things strategic research roadmap*. Internet of Things-Global Technological and Societal Trends, pp.9-52, 2011.

[3] K. Xie, H. Chen and L. Cui. *PMDA: A physical model driven software architecture for Internet of Things*. Computer Research and Development, pp.1185-1197, 2013(In Chinese).

[4] C.A.R. Hoare. *Communicating Sequential Processes*. New Jersey: Prentice-Hall International, 1985.

[5] R.C. Martin. *Agile software development: principles, patterns, and practices*. Prentice Hall PTR, 2003.

[6] K.J. Lieberherr and I.M. Holland. *Assuring good style for object-oriented programs*. Software, pp.38-48, 1989.

[7] D. Garlan, S.W. Cheng, A.C. Huang, et al. *Rainbow: Architecture-based self-adaptation with reusable infrastructure*. Computer, pp.46-54, 2004.

[8] S.W. Cheng. *Rainbow: cost-effective software architecture-based self-adaptation*. ProQuest, 2008.

[9] P. Ranjan and A.K. Misra. *A novel approach of requirements analysis for agent based system*. Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2006.

[10] L. Xiao, D. Robertson, M. Croitoru, et al. *Adaptive agent model: an agent interaction and computation model*. Computer Software and Applications Conference2, COMPSAC, pp.153-158, 2007.

[11] J. D. Farmer, D.Foley. *The economy needs agent-based modelling*. Nature, pp.685-686, 2009.

[12] M. Salehie, L. Tahvildari. *Self-adaptive software: Landscape and research challenges*. ACM Transactions on Autonomous and Adaptive Systems (TAAS), 2009.

[13] R. Rouvoy, P. Barone, Y. Ding, et al. *Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments*. Software engineering for self-adaptive systems. Springer Berlin Heidelberg, pp.164-182, 2009.

[14] A. Kliem, A. Boelke, A. Grohnert, et al. *Self-adaptive middleware for ubiquitous medical device integration. e-Health Networking, Applications and Services*. Healthcom, pp.298-304, 2014.

[15] L. Gasser *Social conceptions of knowledge and action: DAI foundations and open systems semantics*. Artificial intelligence, pp.107-138, 1991.

[16] M. Georgeff, B.Pell, M. Pollack, et al. *The belief-desire-intention model of agency*. Intelligent Agents V: Agents Theories, Architectures, and Languages. Springer Berlin Heidelberg, pp.1-10, 1999.

[17] G. Boella, and L.V.D. Torre. *The ontological properties of social roles in multi-agent systems: Definitional dependence, powers and roles playing roles*. Artificial Intelligence and Law, pp.201-221, 2007.

[18] R.J. Allen. *A Formal Approach to Software Architecture*. Carnegie-mellon univ pittsburgh pa school of computer Science, 1997.

[19] D. Easley,and J. Kleinberg. *Networks, crowds, and markets*. 2012.

[20] A. Belapurkar. CSP for Java programmers. Online, June, 2005.

[21] K. Letsholo, E.V. Chioasca ,and L. Zhao. *An integration framework for multi-perspective business process modeling*. Services Computing (SCC), pp.30-40, 2012.