A Feedback-based Adaptive Service-Oriented Paradigm for the Internet of Things

Yuji Dong[1,2], Kaiyu Wan[2], and Yong Yue[2]

[1] University of Liverpool, Liverpool L69 3BX, UK
[2] Xi'an Jiaotong Liverpool University, Suzhou 215123, China
yuji.dong@liverpool.ac.uk, {kaiyu.wan, yong.yue}@xjtlu.edu.cn

Abstract. With integrating physical devices into digital world, Internet of Things (IoT) have presented tremendous potential in various different application domains such as smart cities, intelligent transportation, smart home, healthcare and industrial automation. However, current IoT solutions and usage scenarios are still very limited because of the difficulty in sensing the context in continuously changing environments and adaptation to the changes accordingly. The complex dynamic interactions between system components and physical environments are a bit challenging especially when there are other concerns such as scalability and heterogeneity. To solve this problem, a novel adaptive service-oriented paradigm is proposed to support IoT from a low-level viewpoint. The paradigm can overcome some disadvantages of REST (Representational State Transfer) architecture style in the IoT. Two classical examples are illustrated using the proposed paradigm by adding an extra constraint based on REST to improve system states verification and enhance the functionality in modelling physical processes.

1 Introduction

IoT are envisioned to integrate the physical world into computer-based systems. Recent years, with the advanced technique development on sensors, networking and data processing etc., IoT have illustrated great potential in various different fields [12]. However, even after decades of research on system aspects of the IoT, developing IoT based systems is still facing many challenges on the high level system requirements like scalability, inter-operability and fault tolerance [19]. Moreover, most of current IoT applications are coping with data collecting and processing without involving many complex physical behaviours, because current IoT solutions and usage scenarios are still very limited in modeling complex behaviours in continuously changing physical environment.

Context adaptation plays an important role in continuously changing physical environment. Recent years, because of the rapid development of mobile computing and big data, there are plenty of context-sensitive data in the IoT systems, therefore the context-awareness in IoT draws a lot of research attention. For example, there are many investigation on context-awareness in models [25], architectures [7] and middlewares [6]. On the other hand, adaptation is more difficult than context-awareness. It is usually solved by constructing the feedback

loop [2] at different abstracting level like architectures [5], behaviour models [4] and frameworks [22].

REST (Representational State Transfer) is a widely used architecture style and also popular in the IoT fields because of its low entry barrier and scalability merits. However, the REST architecture style was particularly designed for distributed hypermedia systems and it sometimes does not fit IoT requirements. In particular, it is difficult for REST to provide complex operations and high level abstraction, while in the IoT systems, the physical behaviours usually need complex behaviour models which REST cannot provide. Therefore two main issues arise, i.e., system states verification and physical behaviours implementation, which we will discuss in more details in the Section 2.

To address these two issues, we propose the Feedback-based Adaptive Service-Oriented Paradigm (FASOP) which can apply at the programming language level to support context adaptation in the IoT systems. Furthermore, the FASOP can be used to add more constraints in order to use the REST style in the IoT systems to overcome these two limitations.

The rest of the paper is organised as follow. Section 2 explains the motivation of the proposed approach. Then the definition and description of the FASOP is presented in Section 3. In Section 4, the FASOP is applied in the REST as an extra constraint. Section 5 illustrates a simple implementation of the FASOP and the two cases discussed before are implemented with the FASOP to express the advantages in Section 6. Finally, Section 7 compares some related works and Section 8 gives the conclusion and future work.

2 Motivation

The REST architecture style is one of the most successful architectures designed for Web applications with the requirements including low entry barrier, extensibility, distributed hypermedia, anarchic scalability and independent deployment [11]. Many features of the REST architecture style can also benefit the IoT system requirements like low entry barrier, decentralization, scalability, robustness and easy deployment. For example, Guinard proposed the resource oriented architecture for the web of things based on REST principles [13]. Furthermore, many existing tools and techniques have supported the REST architecture style, therefore it is easy to integrate with current web technologies. As a result, it has been widely used in many IoT systems such as *Intelligent Buildings* [9], *Smart Homes* [15], *Smart Grids* [17] and *Smart Cities* [20]. Among the 39 available IoT platforms that are surveyed in [18], only 7 platforms do not have REST API.

However, using REST architecture style in the IoT systems may cause two problems in the system states verification and physical behaviours implementation. Below we will use two examples to explain them respectively.

2.1 Issue of System States Verification

To address the issue of system state verification, a scenario in the *Smart Home* is used to explain how the REST style may cause a wrong system states verification.

The scenario is to turn on/off a lamp in a room. Assume there is a controller for a lamp in the room, and it has two operations *switchOn* and *switchOff*. The typical model and design with RESTful interface for this scenario could be as Fig. 1. Based on the HTTP standards, if the response status code is *"200 OK"*, the operation successes, and if the response status code is *"5xx"*, the operation failed with service side error.

However, the problem is that even if the response of the status code *"200 OK"* is obtained, the whole operation cannot guarantee to be successful. The returned *"200 OK"* only means the controller has been successfully triggered, but the lamp may still be off for some unknown reasons, for example, due to the network problem, so the returned status code cannot reflect the real situation.

This kind of problems can be fixed by other fault tolerance mechanisms in middlewares, however, it makes the solution more complex with extra requirements on techniques and tools. Especially, some methodologies may break the constraints in the REST style, and make it more difficult to model the system states and behaviours.

This paper intends to provide a paradigm with feedback mechanism for better system states verification in the IoT systems, so the services developed in the IoT systems, especially in REST style, can be more accurate and reliable.

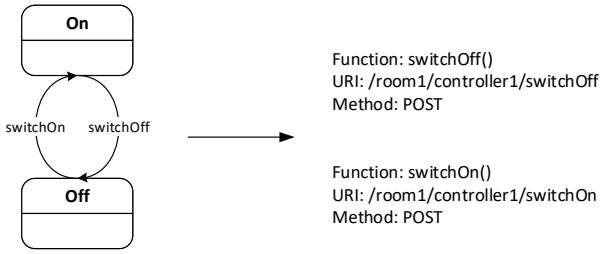

Fig. 1: Model and Design with RESTful interface to Turn On/Off a Lamp

2.2 Issue of Physical Behaviours Implementation

The second issue of physical behaviours implementation is a big problem for developing REST style services in the IoT systems. More specifically, any implementation of continuously physical behaviours with REST style services can be difficult, because the REST style services have limited operations (GET, POST, PUT, DELETE) that cannot fully match the continuously changing physical behaviours. Below we use a scenario of braking a car to explain the limitation.

Assume we need a braking service, which can brake a car based on current conditions and decisions. This scenario cannot be modelled by a simple state

machine. The dynamic physical behaviours of the car can be expressed as follow:

$$\dot{s} = \frac{ds}{dt} = v, \dot{v} = \frac{dv}{dt} = a \tag{1}$$

where s represents the passage within time t with the velocity v, and a is the acceleration. From the REST style services development point of view, we need the GET methods for three variables, s, v and a first, and a POST method to call the *brake service* with parameters a and v and expected passing distance s.

However, it is impossible to ignore all disturbances and uncertainties in the physical environments, so calling a simple *brake service* with parameters a and v and expected passing distance s may cause unpredictable effects, that is, the real passing distance s' is far from s. Furthermore, it is very difficult to map the physical braking device to the braking service because quantitatively describing the action is hard. To overcome the limitations of the open-loop controller, control theory introduces feedback and a closed-loop controller that can use feedback to control states or outputs of a dynamic system, and the Fig 2 indicates the physical behaviour. However, the traditional REST style services cannot perfectly implement this model.

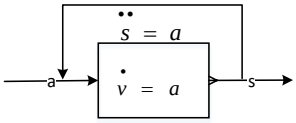

Fig. 2: The Model for Braking Service

The paradigm proposed in this paper can also be used to enhance the functionalities of the REST to support more complex operations in physical world in a natural way, because it fits the mathematical form of calculus.

3 Feedback-based Adaptive Service-Oriented Paradigm

In the SOA based IoT systems, we distinguish three types of different services, i.e., *Virtual Service*, *Perceptive Service* and *Actuating Service*. The three types are evaluated by the interactive patterns from the service providers to the physical environment.

Virtual Service Most of traditional software services are virtual services with no interaction with physical environment. Even in IoT systems like smart home, most of services are still virtual services which can store temperature data or convert temperature from one unit to another, for example.

Perceptive Service Perceptive Services are usually provided by sensors and responsible for detecting the physical environment. The perception provided from the perceptive service can be temperature, pressure or vision etc.

Actuating Service Actuating Services are expressed as services executing real actions in the physical environment. For example, in smart home, turning on/off a light or air condition are actuating services.

An interface I of a service SP, denoted by I_s is defined by a signature and a behavioral model. In the IoT, for any given *Actuating Service*, its interface I_{ac} can be specified by *context*, *signature* and *behavioral model*. *Context* defines information depending on service requesters and service environment. *Signature* corresponds to operation profiles provided by the actuating service. *Behavioral model* is represented by Petri nets to describe the adaptive pattern.

Definition 1 (Context). *We define the context as a typed relation [24], a set of ordered pairs of* (d, x) *where* d *is a dimension,* T_d *is the type of* d *and* $x : T_d$*. Let* D *denote the set of all possible dimensions, and* $T = T_d | d \in D$ *be the set of types associated with the dimensions. A context* c *is a finite relation* $\{f(d, x) | d \in D \wedge x : T_d\}$*. The degree of the context* c *is* $|dom\ c|$*.*

Definition 2 (Signature). *A Signature is a set of operation profiles. An operation profile is the description of an operation containing the name of an operation, with its argument types and its return type. For the actuating service interface* I_{ac}*, its signature is defined by a tuple* $< O^{as}, O^{ps}, \Gamma >$*, where* O^{as} *is a set of operation profiles provided by the actuating service and* O^{ps} *is a set of dependent operation profiles provided by other perceptive services.* Γ *is the function* $\Gamma : O^{as} \rightarrow O^{ps}$*. For any single operation profile* $o_{as} \in O^{as}$*, it has a set of callable operations from other perceptive services* $O^{ps'} \subseteq O^{ps}$ *and* $O^{ps'} \neq \emptyset$*, which is defined as* $\gamma \in \Gamma : o_{as} \rightarrow O^{ps'}$*,* $O^{ps'} \neq \emptyset$*.*

Definition 3 (Behavioral model). *The behaviour in the service can be modelled as a Petri net* $SN =< P, T, F, i, o >$*, where* P *and* T *are disjoint sets of* places *and* transitions*. Places represent states that contain tokens with multiple attributes, and transitions represent activities that can be guarded; transitions are fired when all the tokens in the corresponding input places arrive. Places and transitions are connected through arcs.*

Definition 4 (Service Interface). *A service interface is a tuple* $< CP, S, B >$*, where:* CP *is a context profile, and* S *is a signature with its corresponding behaviour model* B*.*

The behavioural model of FASOP is represented as a Petri net to indicate the atomic operation in *Actuating Services*. As shown in the left side of Figure 3, before applying the paradigm, t_1 is a transition provided by the *Actuating Service* and p_1, p_2 are pre-condition and post-conditions of t_1 respectively. From the process point of view, if the operation in t_1 is a function call $< result : func(params...) >$, then p_1 is to map the function name $func$ and

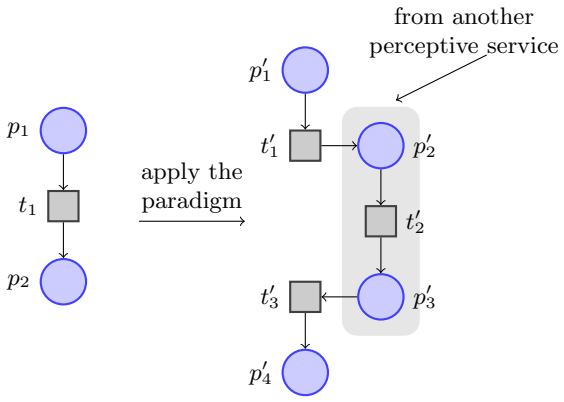

Fig. 3: the Petri Net Behavioural Model for an Actuating Service

parameters ($params...$), and p_2 is to check the return value $result$. However, if the $< result : func(params...) >$ has any action in physical environment, it is nearly impossible to guarantee all post-conditions from the programming language level, because the post-conditions of t_1 may contain some physical effects that cannot be detected by the *Actuating Service* itself.

In our approach, the proposed paradigm is a mechanism with feedback mechanism to solve this problem at the programming language level. Feedback control is a central element of control theory, and the importance in self-adaptive systems has already been discussed in [2].

Among the three types of the services in IoT, the feedback loop can be constructed by *Actuating Services* and *Perceptive Services*. A service signature explicitly exposed by a *Actuating Service* is a set of operations that need to declare reachable *Perceptive Services* with specific operations. For a single operation profile, it is expressed as shown in Table 1. Then, any service call to $funcAS$ has to pass all required parameters including context information and at least one extra service call as an available *Perceptive Service*. The behavioural model is at the right side of the Figure 3. The service call at t_1 is changed from $< result : func(params...) >$ to $< result : func(params..., PS.funcPS, t) >$, which contains another function $funcPS$ from another *Perspective Service* and the latency time t which is the waiting time to get the feedback perceptions. In this way, the verification for post-conditions is more reasonable, since the post-conditions with physical properties can be verified through the perceptions of the physical environment by the *Perceptive Services*. With this new paradigm, the *place* p'_2 can verify whether the operation $func$ is operated successfully and *place* p'_4 can eventually check if the operation $func$ has desired behaviours based on the perceptions. In Figure 4, a sequence diagram shows the detailed processes from the implementation perspective.

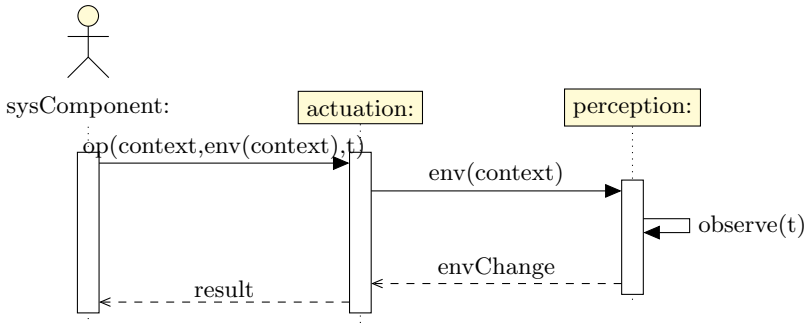

Fig. 4: Sequence Diagram for calling a Actuating Service

Table 1: A Single Operation Format in Actuating Service

Operation Name	$funcAS$
Parameters	$p_1, p_2, ..., p_n$
Available Operations	$PS_1.op1, PS_2.op_1, PS_2.op_2, ..., PS_i.op_j$

4 Extending REST for the IoT based on FASOP

In [10], an example of using REST-based architecture server to control a robot is presented. The author concluded that REST sometimes is inconvenient compared to other RPC style web services because it does not have any functionalities like callbacks to support complex modelling with the states. The key problem is that keeping all required information in a single request to model physical behaviour while keeping stateless interactions is difficult. In physical environment, most of the continuously physical behaviours are modelled based on differential equations so it needs at least two states to express a continuous physical behaviour. Therefore at least two states in the response are needed to model the physical behaviours in any single request. With the FASOP, any single service call to an *Actuating Service* actually becomes a transaction, so all required information can be wrapped up to model physical behaviours within one request. This paradigm can be simply converted to an extra constraint for the REST and the new constraint is expressed as follow:

– Any operation from the *Actuating Services* has to operate in a complete feedback loop containing the perception to physical environment and the response need to have at least two states of the requested entity.

The high-level model we use for IoT systems is based on [16], which is defined as a tuple $RS = \langle R, I, B, \eta, C, D, \sim, OPS, RETS \rangle$, where R is a set of resources; I is a set of resource identifiers; $B \subseteq I$ is a finite set of root identifiers; $\eta : I \to R$ is a naming function, mapping identifiers to resources. C is a set of client identifiers and D is a set of data values, with an equivalence relation $\sim \subseteq (D \times D)$; OPS is a finite set of methods; and $RETS$ is a finite set of return codes. The detailed

model is similar as the model provided by [21], where the only difference is for modelling services for actuators.

Resource identifiers are modelled as URIs, represented as the following scheme:

$$URI = scheme : [//host[: port]][/]path[?query][\#fragment]$$

The descriptions of a service can be obtained by sending a GET to a particular resource via *URI*. Most of the service calls are at protocol level via message delivery and the main difference is on the *Actuating Service* calling. The request to an *Actuating Service* needs to contain at least one available *Perspective Service* operation.

5 Implementation Methods

In this section, we will use Java web service to express a simple implementation of this paradigm.

Based on the former definition in section3, the services in the IoT systems are concluded as: *Virtual Service*, *Perceptive Service* and *Actuating Service*. Since *Virtual Service* is just normal web service, we develop two extra interfaces: *PerceptiveService* and *ActuatingService*. To hide many network details, we assume all the services can remotely call another service from a different device based on RPC framework or Actor model [1]. Fig 5 indicates a basic example of these two interfaces.

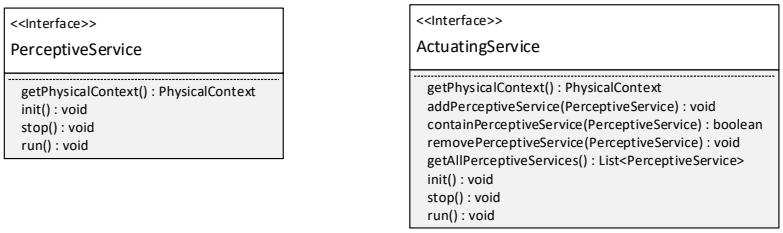

Fig. 5: The Basic Class Diagram of the two Physical Interfaces

The main purpose of the interface design is to do type checking in the development. By using annotation in Java, we can restrict the developer to include at least one *PerceptiveService* as a parameter in any *Actuating Service* annotated by @WithFeedback. However, the type checking at this level needs to remotely call a method, if you want to use the JAX_RS (Java API for RESTful Web Services)[14] standard to develop REST style services, the parameters are all String type for the services mapped from the URI, thus you cannot do type checking to confirm the *PerceptiveService* as a parameter. In this case, you need to check the *PerceptiveService* in the function of the service.

The implementation is only a lightweight version of the FASOP implementation, because we need to extend the HTTP or CoAP (Constrained Application Protocol) to fully support the FASOP, which is considered as a part of future work.

6 Case Studies

In this section, we use the two examples in the motivation section to illustrate the advantages of the FASOP.

6.1 Turn on/off a Lamp in the Smart Home

For the scenario in the Smart Home to turn on/off a lamp in a room, the issue is that the response status code cannot express the correct system status. To solve this problem, we use the FASOP to modify the original approach and the changes are as follow:
Function: switchOn() → switchOn(PerspectiveService,t)
URI: /room1/controller1/switchOn
→ /room1/controller1/switchOn/?perceptiveservice=lightsensor&time=1s
Method: POST
The implementation details are expressed in Fig 6. In this implementation, the successful status code correctly reflects a guaranteed successful confirmation.

```
public class LampService implement ActuatingService{
        ......
        @POST
        @Path("/room1/controller1/switchOn/{perceptiveservice}\{time}")
        public String switchOn(@PathParam("perceptiveservice") String p,@PathParam("time") String time){
                if(!PerceptiveServices.containskey(p)){
                        switchOn();
                        return "No PerceptiveService Found, Switched On";
                } else {        switchOn();
                        Thread.sleep(Integer.parseInt(time)*1000);
                        String state = PerceptiveServices.get(p).getResponse();
                        if(state.equals("On")){
                                return "Switched On, Successfully";
                        } else{  return "Failed";  } } }
        ......
}
```

Fig. 6: Implementation Sample of using FASOP to turn on a Lamp

6.2 Brake an AutoDriving Car

Compared to the traditional REST style development, the FASOP can help to transfer the physical behaviour model to software development in a more natural

way. Below we use the example introduced in Section 2 to explain how the FASOP can help to transfer physical behaviour model to software development.

Based on the Equation 1, in a very short time $\Delta t = t' - t$, we have the following form of the equation:

$$\dot{s} = \frac{ds}{dt} = \frac{s' - s}{t' - t} = v, \dot{v} = \frac{dv}{dt} = \frac{v' - v}{t' - t} = a = \frac{s' - s}{(t' - t)^2}$$

With the traditional REST style service development, it is very difficult to quantitatively evaluate and analyze acceleration. However, the FASOP can fit the closed-feedback model, thus the exact acceleration value can be easily evaluated via the distance and time. Furthermore, we can continuously change the acceleration via braking physically and all effect can be evaluated though the service in real-time. The function of this braking service can be as follow:
Function: braking(PerspectiveService,t)
URI: /car/brake/braking/?perceptiveservice=distancesensor&time=1ms
Method: POST
In any moment, with this braking service, we can also predict the future passing distance s_f during the time period t_f. The predication can be based on: $s = v * t - \frac{a * t^2}{2}$ if the acceleration keeps the same.

7 Related Works

There are many researches on the development of the IoT systems to support context-awareness and adaptation. In [23], a platform is developed as ContextServ to simplify the development of context-aware Web services adopting high-level modelling language. In [3], a design for adaptation approach is proposed to support the development, deployment and execution of systems in dynamic environments by exploiting service refinement and re-configuration techniques. In [22], the MAPE-K feedback loop is used to support a synchronization and adaptation mechanism for real world process as a process-based framework. It uses a different perspective from combining processes' virtual world and real world effects to build self-adaptive IoT systems. The work can achieve a high level of autonomy and resilience against failures for physical world process. In [8], the authors provide the methodology of using model-based service oriented architecture with service composition to support self-adaptation. The work is solid and also provides fault tolerance mechanism. In [4], the service adaptation is achieved using service composition for automatic reconfiguration based on the rich interface specifications. Following this idea, they used the Discrete Time Markov Chains in a language to describe the impact of adaptation tactics and the assumption about the environment.

Our approach is a paradigm that can be used in the current service based technologies, especially for a widely used style of REST. Because of the special constraints of the REST style, the REST style services are not very suitable for context-adaptation in the IoT system. Compared to others' work, we used a

different perspective of designing the Feedback-based Adaptive Service-Oriented Paradigm to support context-adaptation in service development, especially for REST style service development which rarely supported the context-adaptation. Furthermore, we proved that this paradigm can overcome two issues in using REST style services in the IoT systems.

8 Conclusion and Future Work

The two issued caused by using the REST style services in the IoT systems are from the lack of developing the complex behaviour models in the REST style. To overcome the problem, we proposed the Feedback-based Adaptive Service-Oriented Paradigm to provide the context-adaptation ability at low level for service development, therefore the REST style services can implement complex behaviour processes based on the context-adaptation.

The implementation in this paper is a simplified version to use the current web technologies. To fully implement the FASOP in the REST style, we plan to develop a protocol by extending the HTTP or CoAP (Constrained Application Protocol) based on REST model. In addition, in the future we will also develop and deploy this paradigm in some real IoT systems.

References

1. Agha, G.A.: Actors: A model of concurrent computation in distributed systems. Tech. rep., MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL INTELLIGENCE LAB (1985)
2. Brun, Y., Serugendo, G.D.M., Gacek, C., Giese, H., Kienle, H., Litoiu, M., Müller, H., Pezzè, M., Shaw, M.: Engineering self-adaptive systems through feedback loops. In: Software engineering for self-adaptive systems, pp. 48–70. Springer (2009)
3. Bucchiarone, A., De Sanctis, M., Marconi, A., Pistore, M., Traverso, P.: Design for adaptation of distributed service-based systems. In: International Conference on Service-Oriented Computing. pp. 383–393. Springer (2015)
4. Camara, J., Canal, C., Salaün, G.: Behavioural self-adaptation of services in ubiquitous computing environments. SEAMS 9, 28–37 (2009)
5. Cámara, J., Lopes, A., Garlan, D., Schmerl, B.: Adaptation impact and environment models for architecture-based self-adaptive systems. Science of Computer Programming 127, 50–75 (2016)
6. Caporuscio, M., Raverdy, P.G., Issarny, V.: ubisoap: A service-oriented middleware for ubiquitous networking. IEEE Transactions on Services Computing 5(1), 86–98 (2012)
7. Chen, H., Finin, T., Joshi, A., Kagal, L., Perich, F., Chakraborty, D.: Intelligent agents meet the semantic web in smart spaces. IEEE Internet Computing 8(6), 69–79 (2004)
8. Cubo, J., Canal, C., Pimentel, E.: Model-based dependable composition of self-adaptive systems. Informatica 35, 51–62 (2011)
9. Dinh, N.T., Kim, Y.: Restful architecture of wireless sensor network for building management system. KSII Transactions on Internet and Information Systems (TIIS) 6(1), 46–63 (2012)

10. Esteller-Curto, R., Cervera, E., del Pobil, A.P., Marin, R.: Proposal of a rest-based architecture server to control a robot. In: Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on. pp. 708–710 (July 2012)
11. Fielding, R.T.: Architectural styles and the design of network-based software architectures. Ph.D. thesis, University of California, Irvine (2000)
12. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of things (iot): A vision, architectural elements, and future directions. Future generation computer systems 29(7), 1645–1660 (2013)
13. Guinard, D., Trifa, V., Wilde, E.: A resource oriented architecture for the web of things. In: Internet of Things (IOT), 2010. pp. 1–8 (Nov 2010)
14. Hadley, M., Sandoz, P.: Jax-rs: Java api for restful web services. Java Specification Request (JSR) 311 (2009)
15. Kim, S., Hong, J.Y., Kim, S., Kim, S.H., Kim, J.H., Chun, J.: Restful design and implementation of smart appliances for smart home. In: Ubiquitous Intelligence and Computing, 2014 IEEE 11th Intl Conf on and IEEE 11th Intl Conf on and Autonomic and Trusted Computing, and IEEE 14th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UTC-ATC-ScalCom). pp. 717–722. IEEE (2014)
16. Klein, U., Namjoshi, K.S.: Formalization and automated verification of restful behavior. In: International Conference on Computer Aided Verification. pp. 541–556. Springer (2011)
17. Meloni, A., Atzori, L.: A cloud-based and restful internet of things platform to foster smart grid technologies integration and re-usability. In: Communications Workshops (ICC), 2016 IEEE International Conference on. pp. 387–392. IEEE (2016)
18. Mineraud, J., Mazhelis, O., Su, X., Tarkoma, S.: A gap analysis of internet-of-things platforms. Computer Communications 89, 5–16 (2016)
19. Mukhopadhyay, S.C., Suryadevara, N.: Internet of things: Challenges and opportunities. In: Internet of Things, pp. 1–17. Springer (2014)
20. Paganelli, F., Turchi, S., Giuli, D.: A web of things framework for restful applications and its experimentation in a smart city. IEEE Systems Journal 10(4), 1412–1423 (2016)
21. Prehofer, C.: Models at rest or modelling restful interfaces for the internet of things. In: Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on. pp. 251–255. IEEE (2015)
22. Seiger, R., Huber, S., Heisig, P., Assmann, U.: Enabling self-adaptive workflows for cyber-physical systems. In: International Workshop on Business Process Modeling, Development and Support. pp. 3–17. Springer (2016)
23. Sheng, Q.Z., Pohlenz, S., Yu, J., Wong, H.S., Ngu, A.H.H., Maamar, Z.: Contextserv: A platform for rapid and flexible development of context-aware web services. In: Proceedings of the 31st International Conference on Software Engineering. pp. 619–622. ICSE '09, IEEE Computer Society, Washington, DC, USA (2009), http://dx.doi.org/10.1109/ICSE.2009.5070570
24. Wan, K.: A brief history of context. In: International Journal of Computer Science Issues, Volume 6, Issue. Citeseer (2009)
25. Yanwei, S., Guangzhou, Z., Haitao, P.: Research on the context model of intelligent interaction system in the internet of things. In: IT in Medicine and Education (ITME), 2011 International Symposium on. vol. 2, pp. 379–382. IEEE (2011)