A Generic Solution for IoT Ontology Model Based on OCF Standard

Peili Shi[1], Yunlong Tian[2,4(✉)], Li Niu[3,4], Xiaoran Ma[2,4], Yuanzhen Ge[1], and Yonghua Li[1]

[1] School of Information and Communication Engineering, Beijing University of Posts and Telecommunications (BUPT), Beijing 100876, China
{2019140105,geyuanzhen,liyonghua}@bupt.edu.cn
[2] Qingdao Haier Technology Co., Ltd., Qingdao, China
{tianyl,maxiaoran}@haier.com
[3] Qingdao Haier Intelligent Home Appliance Technology Co., Ltd., Qingdao, China
niuli.dpg@haier.com
[4] National Engineering Research Center for Digital Home Network, Qingdao, China

Abstract. In the ever-evolving landscape of the Internet of Things (IoT) ecosystem, the proliferation of diverse standards has posed challenges in achieving seamless interoperability among devices, platforms, and systems. A critical concern within IoT is semantic interoperability, where differing semantics hinder effective communication between devices and systems. Various IoT consortiums have proposed solutions such as middleware technologies for protocol bridging and network semantic technologies for unified models. However, complete interconnection remains a challenge. In response, this paper introduces the Thing Specification (ThingSpec) IoT ontology model, designed to address semantic and syntactic discrepancies by integrating multiple standard ontologies. The ThingSpec model distills commonly used concepts, including devices, profiles, properties, actions, events, and services, while enhancing service collections and management. The model is organized into four functional components: the thing model, static model, dynamic model, and collection model, catering not only to device functionality but also facilitating multi-device interactions. By implementing the model within the Open Connectivity Foundation (OCF) framework, we demonstrate its ability to dynamically link actions, events, and services in specific scenarios. Evaluation results indicate the model's prowess in integrating multiple standard ontologies and its robust language completeness and extensibility.

Keywords: Internet of Things (IoT) · Open Connectivity Foundation (OCF) · Ontology · RESTful

1 Introduction

The Internet of Things (IoT) refers to the interconnection of all things through Internet technology, which transforms traditional usage into intelligent driven

Supported by National Key R&D Program of China No. 2022YFC3803204.

F. Gao et al. (Eds.): Chinacom 2023, LNICST 590, pp. 280–299, 2024.
https://doi.org/10.1007/978-3-031-67162-3_18

[1]. As IoT devices are increasingly incorporated into applications, a number of issues arise, especially the interoperability of IoT. The interoperability of IoT is defined by IEEE as "the ability of two or more systems or components to exchange information and to use the information that has been exchanged" [2]. Nowadays, most of the current IoT devices manufactured follow a closed protocol model. Hence, the interoperability across manufacturers is difficult to achieve without a unified standard. To address the interoperability barrier of heterogeneous devices, most of the literature adopted the intervention solution at the application layer, including providing a common API, designing a gateway to convert multiple protocols, etc. [3–8]. However, these solutions are limited in scope and cannot be a general model, so they do not solve the interoperability problem well.

The interoperability of IoT mainly considers the following aspects: technology, syntax, semantics, and organization [9]. One of the most pressing topics is semantic interoperability. Semantics refers to a set of information models and ontologies to unify the terminology used for data exchange [10]. The problem of semantic interoperability refers to how to describe the "things" in the IoT with a unified set of semantic technologies. To achieve semantic interoperability, Amit Kr Mandal et al. designed a service model, covering the physical properties of things and related contextual content, and implemented a REST-based weather monitoring service [11]. Oscar Novo et al. introduced an extension scheme based on the WoT architecture proposed by W3C, intervening in auxiliary translation between W3C and IETF standards, so as to integrate into the two standards, and use a combination of quantitative and qualitative evaluation methods to verify the feasibility and scalability of this scheme [12].

On the one hand, the proposal of the new model has not been systematically implemented, which makes it impossible to prove its feasibility. On the other hand, the optimization based on the existing model is mostly limited between the two standards, which has a certain gap with the general model. To address the above problems, the main work of this paper is summarized as follows.

1. An ontology-based Thing Specification (ThingSpec) model is proposed, which is divided into thing model, static model, dynamic model and collection model, constructed with the corresponding classes and individuals.
2. An implementation process of ThingSpec model is designed based on the Open Connectivity Foundation (OCF) framework, which combines static model and dynamic model to realize the action, event, service, and multi-device linkage communication process.
3. An evaluation of the ThingSpec model is completed according to the metrics of the ontology to verify the comprehensive performance of the model.

In the remainder of the article, Sect. 2 introduces the related work. Section 3 details the ontology-based ThingSpec Model. Section 4 presents the implementation process of the ThingSpec model. Section 5 analyzes the ontology performance of the ThingSpec model. Finally, the overall work is summarized, and the improvements in the future are pointed out.

2 Related Work

This section will be divided into two parts: ontology of the IoT, OCF framework.

2.1 Ontology of the IoT

In order to solve the semantic problems in various fields in the IoT, there are already more than 380 ontology-based IoT projects [13]. The ontology is a set of concepts and categories in a specific domain that explicitly describe the relationships between them [14]. Its main characteristics as a knowledge representation are flexible, clearable, sharable, and reusable [15]. The ontologies to be introduced next are the Sensor Observation Sample Actuation (SOSA) ontology, the Smart Appliances Reference (SAREF) ontology, and the Time ontology. Also, the thing description of the Web of Things (WoT) proposed by World Wide Web Consortium (W3C) will be introduced.

SOSA Ontology. SOSA ontology divides sensors into three types, like observation, actuation, and sampling [16].

- As an observation sensor, when the sensor enters the observing state, it will start observing the observable properties until the result equals the ideal value.
- An actuation sensor mainly contains an actuation module, which is composed of four main classes: actuation, actuator, actuatable property, and feature of interest. The actuator generates the actuation, the actuation triggers the actuatable property, and the feature of interest is the ideal value of the property that the actuator wants to achieve after the actuation.
- A sampling sensor mainly contains a sampling module, which is irrelevant to the model proposed in this paper, so it will not be repeated here.

SAREF Ontology. SAREF ontology focuses on knowledge construction in the field of smart home appliances. Among them, a device is defined as a tangible object used in a home, public building, or office to accomplish a specific task. To accomplish this task, the device performs one or more functions. A functional group that is exposed outside the device and can be discovered and controlled by other devices, called a service, is a representation of a group of functions. In addition to the functional information, to supplement other information of the device, it also has a profile module [17].

Time Ontology. Time ontology is to express qualitative time information in professional terms. It takes a temporal entity as the starting parent class and expands two subclasses into instant and interval. The interval includes the subclass proper interval, which includes the date time interval [18].

Thing Description of the WoT. The thing description provided by WoT represents the metadata and interface of the thing. In the interaction model of the thing description, things have three main interaction elements: property, action, and event. The property describes a property-based interaction model for obtaining and controlling the current state of things. The action describes the process of physical scheduling, including the triggering of physical sensors or the invocation of abstract platforms, such as interactions with cloud platforms. The event is used to push information. These three types of elements are inherited from the interaction model to define the interaction interface in a unified format [19].

On the application of the ontology, the reuse of multiple existing ontologies, combined with the concepts of strategy, context, service, and monitoring, built a comprehensive ontology for Internet of Things (COIoT), which has a wide coverage but does not involve evaluation criteria and cannot verify the feasibility [20]. The semantic Web of Things (SWoT) ontology, combined things-based static descriptions and dynamic interactions following the PAE paradigm (property, action, event), implemented with the Cocktail framework, and evaluated the ontology using performance metrics [21]. However, the SWoT ontology does not involve the concept of service, and has low reusability.

2.2 OCF Framework

In the OCF protocol proposed by the Open Connectivity Foundation, all entities existing in the IoT are mapped as resources, as shown in Fig. 1. The architecture consists of three parts: Resource model, Representational State Transfer (RESTful) operations, and abstractions. A device contains multiple resources, and each resource is constructed based on the resource model. The OCF abstraction is to convert RESTful operations of resources into communication operations in data protocols through mapping [22].

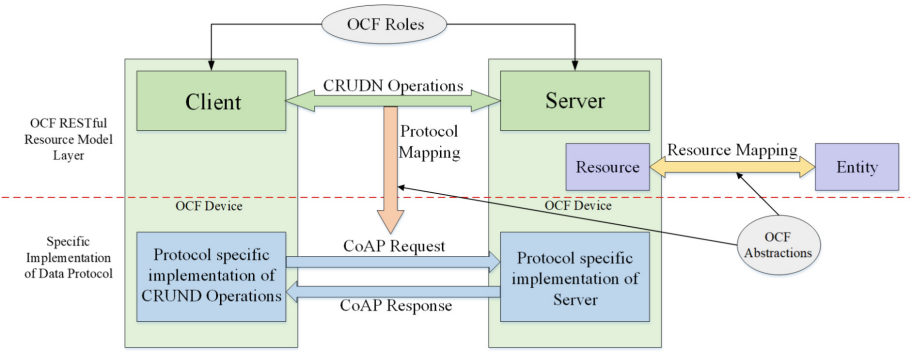

Fig. 1. The architecture of OCF standard.

In OCF's architecture, RESTful operations are also considered CRUDN operations, which are composed of CREATE, RETRIEVE, UPDATE, DELETE, and NOTIFY. Through the protocol mapping, CRUDN operations can be converted into standard HTTP methods, including GET, PUT, POST, and DELETE. GET is used to retrieve resources; PUT and POST are used to create resources and update the status of some resources; DELETE is used to delete resources [23,24]. RESTful architecture is considered a good choice for IoT architecture [24–26].

The Constrained Application Protocol (CoAP) is a communication protocol applied to restricted devices. In the data transmission of the dynamic network, CoAP outperforms HTTP in terms of transmission rate, delay, and overhead [27]. The Internet Engineering Task Force (IETF) proposed a mapping guide from HTTP to CoAP, which enables HTTP clients to obtain resources from CoAP servers through a network proxy [28]. Especially in the IoT, CoAP has been proposed as an alternative to HTTP, playing a similar role, such as the implementation of the OCF standard.

3 Thing Specification Model

The relationship between the core concepts of the ThingSpec model is shown in Fig. 2. Each module is equivalent to a class in the ontology, and the relationship between the modules is the object property in the ontology. The ontology diagram of the ThingSpec model is constructed by protégé, as shown in Fig. 3. According to functional categories, the ThingSpec model (TS) can be divided into: things (T), static models (S), dynamic models (D) and collections (C), as follows:

$$TS = \{T, S, D, C\}. \tag{1}$$

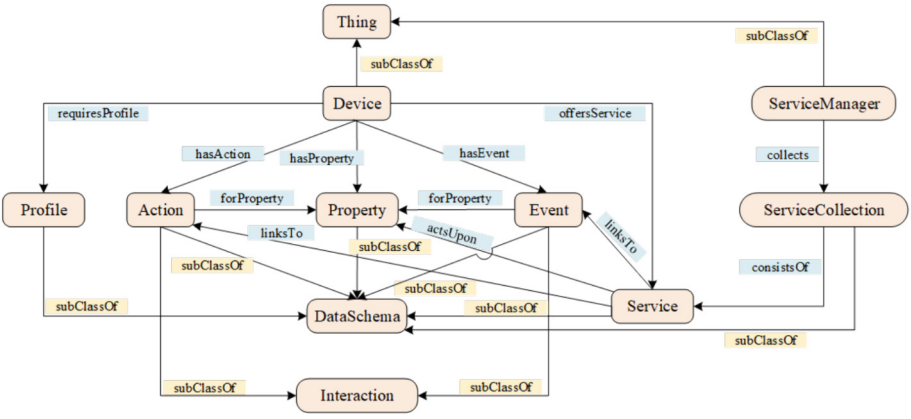

Fig. 2. The overall of the Thing Specification Model.

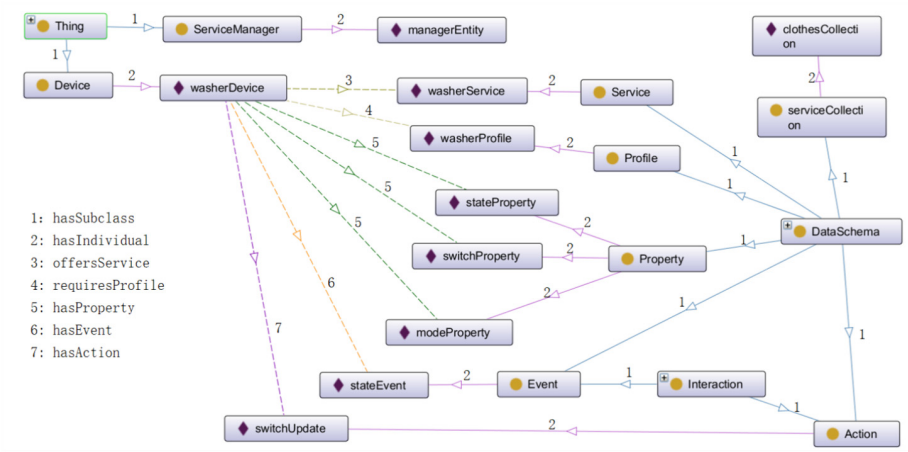

Fig. 3. The ontology diagram of the Thing Specification Model.

3.1 Thing Model

The thing model describes entities that have a certain function in the IoT, including physical entities and virtual entities. It contains a large number of devices (Dev) and a unique service manager (SM), as

$$\{Dev_1, Dev_2, Dev_3, \ldots\} \cup \{X \mid X = SM\} \subset T \tag{2}$$

where Dev_1, Dev_2, Dev_3 are different devices, distinguished by identifiers, which may belong to the same category, such as bedroom lights, living room lights, study lights; or different categories, such as air conditioners, TVs, washing machines; or Hybrid, for example, bedroom lights, living room lights, air conditioners. In a certain area, there will be a large number of devices. The service manager is to manage all service collections in this area. It is mainly used for communication with the device.

Devices have independent and complete systems, most of which are physical entities. As defined by the ISO/IEC standard, it is "the entity of an IoT system that interacts and communicates with the physical world through sensing or actuating" [29]. A device consists of one profile, multiple properties and multiple actions, and may contain multiple events and multiple services at the same time, as

$$\begin{aligned} \{X \mid X = \mathrm{Dev}\} = \{X \mid X = \ \mathrm{Prof}\ \} \wedge \{\mathrm{Prop}_1, \mathrm{Prop}_2, \mathrm{Prop}_3, \ldots\} \wedge \\ \{A_1, A_2, A_3, \ldots\} \wedge (\{E_1, E_2, E_3, \ldots\} \vee \{\mathrm{Ser}_1, \mathrm{Ser}_2, \mathrm{Ser}_3, \ldots\} \vee \emptyset) \end{aligned} \tag{3}$$

where profile, property, and action are necessary elements, and event and service are optional elements which are defined by the device type. In addition, there is a special kind of device, that is, called a client. A client has a profile to describe the basic information, contains zero or more properties to represent the list of

devices which can be controlled, and covers methods for interacting with other devices.

At the data level, profile, property, action, event and service are all described based on the parameters of the data model, so they are all subclasses of the data model. The concept of data model is derived from the thing description of the WoT.

3.2 Static Model

The static model contains a large number of profiles(Prof) and properties(Prop), as follows:

$$\{\mathrm{Prof}_1, \mathrm{Prof}_2, \mathrm{Prof}_3, \ldots\} \cup \{\mathrm{Prop}_1, \mathrm{Prop}_2, \mathrm{Prop}_3, \ldots\} \subset \mathrm{S} \tag{4}$$

where $Prof_1, Prof_2, Prof_3$ are identified as different profiles and assigned to different devices. Each profile contains the basic information of the device, which will not change during the interaction process, and there is no interaction process. Here, $Prop_1, Prop_2, Prop_3$ are different properties, which can exist in different devices or the same device. Each property stores the current status of the device, which is irrelated to dynamic interaction process.

The profile is the integration of static information about the device except properties. It supplements the basic information about the device, including identifier, key, name, description, endpoint, version number, and manufacturer, on the basis of reusing the relevant concepts of the profile form the SAREF ontology.

The establishment of properties is designed to represent the current status of the device and possible status. For example, a washing machine has an operation property, and the current operation state is "laundry in progress". It has a similar concept with the property in the thing description of the WoT, but the difference is that the property proposed by the thing description belongs to the interaction model, and the property in the ThingSpec model only has current static information, and there is no interaction process. In addition to inheriting the data attributes of the data model, it also includes URI, key, state value, observable, mandatory.

3.3 Dynamic Model

The dynamic model contains various actions(A) and events(E), as follows:

$$\{A_1, A_2, A_3, \ldots\} \cup \{E_1, E_2, E_3, \ldots\} \subset \mathrm{D} \tag{5}$$

where A_1, A_2, A_3 are different actions, which can exist in the same device or from different devices. The invocation of each action will generate interactive behavior, which can be captured in the network. E_1, E_2, E_3 are events from the same device or different devices, and their triggering can also cause a certain number of network behaviors.

Actions define behaviors that can control physical states with related behavioral parameters, including parameters required for communication between devices. It is a dynamic communication process that is the same as the action proposed in the thing description of the WoT. Not only that, it also integrates the instant and interval of time ontology. As shown in Fig. 4, actions are divided into four categories: CREATE, DELETE, UPDATE, and RETRIEVE, which are collectively called CRUD operations and are implemented based on the OCF standard.

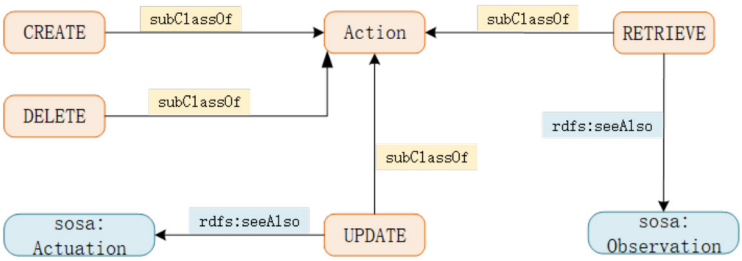

Fig. 4. The classification of Action.

Among them, the update action is related to the actuation of the SOSA ontology, that is, when the action type is update, it means that the physical state of a sensor in the corresponding device needs to be changed, which corresponds to the corresponding operation of the actuation. Likewise, the retrieve action is related to the observation of the SOSA ontology.

The event refers to the interaction model that is triggered and sent to the user after the result value reaches an ideal state in the process of observing a certain property by the device. The difference from the action is that the event is triggered by the device and then reported to the user for information push, while the action is triggered by the user and sent to the device; the event needs to set trigger conditions, which will be automatically triggered, but the action is a human trigger.

3.4 Collection Model

In the definition of the OCF standard, a collection is a centralized representation with linked properties that can bring multiple resources together. The collection contains a large number of services(Ser) and service collections(SC), and a unique service manager, as follows:

$$\{\mathrm{Ser}_1, \mathrm{Ser}_2, \mathrm{Ser}_3, \ldots\} \cup \{SC_1, SC_2, SC_3, \ldots\} \cup \{X \mid X = SM\} \subset C \quad (6)$$

where Ser_1, Ser_2, Ser_3 describe various services from the same or different devices, which are a kind of collection because they contain references to properties, actions or events. Here, SC_1, SC_2, SC_3 are different service collections,

which contain references to different services. The service manager integrates multiple service collections, and naturally includes references to service collections.

The service is exposed outside the device and is used for linkage between different devices, such as laundry service. In the ISO/IEC standard, it is defined as "a distinct part of the functionality that is provided by an entity through interfaces" [29]. It is an independent and complete function group to provide the representation of a group of functions, which is the same as the service from the SAREF ontology. Also, it can be discovered and invoked by other devices. It contains two parameters: service identifier and device identifier. The service identifier is used to expose the service to the network, and the device identifier is used to link the device information of this service. At the same time, each type of service will be executed based on a property. According to requirements, it is chosen whether to link to action or event, and the actual function of the service is defined according to the link.

The service collection combines multiple services from different devices, stores these services according to the settings of different scenarios, and triggers the linkage effect between different services with a certain rule. The variables it contains are: URI, name, key, input list, input on, action list, action on, start.

The service manager is a virtual entity. Its role is to integrate multiple service collections, centrally manage and schedule services, which is an indispensable part of multi-device linkage scenarios. It forms a linkage model with service collections and services, as shown in Fig. 5.

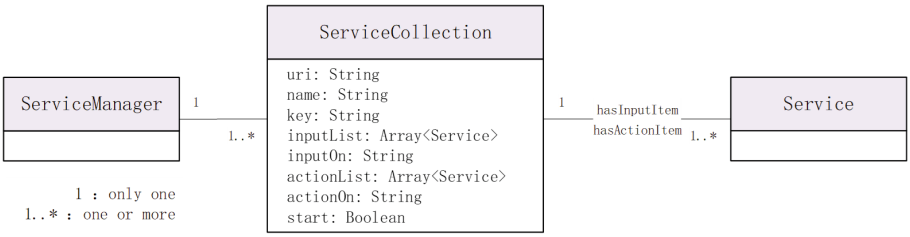

Fig. 5. The UML of the linkage model.

4 System Implementation

This paper takes the smart balcony series of smart homes as an example and implements the system by building multiple virtual devices on the OCF platform to imitate the actual interaction behavior. This scene mainly includes two devices: washing machine and drying hanger, and a service manager. The washing machine includes a switch to activate the washing operation, a mode to select the washing mode including normal wash, quick wash or spin dry, and an operating state to indicate the current washing progress. The drying hanger includes a lifting state, which can be changed to make the drying hanger automatically rise and fall, and a mode for switching between the normal mode and

the disinfection mode. The service manager is mainly the management of multi-device linkage scenarios and integrates the specific services of each device to realize the intelligent linkage mode.

The client interface presents a list of devices from the server, all of which have their own unique identifiers, as shown in Fig. 6.

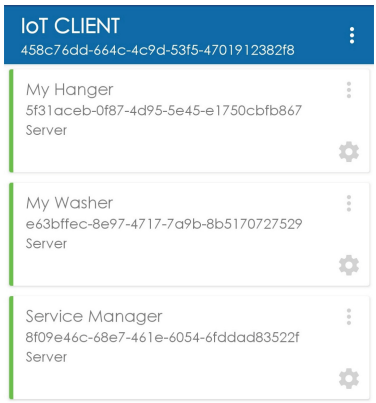

Fig. 6. Device List.

4.1 Ontology Mapping

The first step in the system implementation is to map the ontology model to the OCF framework. A partial Ontology Web Language (OWL) description of the ThingSpec ontology is shown in Fig. 7. In the description, an individual named "washerDevice" has an object property relationship with seven individuals that can correspond to methods in the Java implementation. An individual named "washerProfile" has data properties that can be implemented through creating a new object, setting a value, etc. in a Java class. In a word, the main purpose of ontology mapping is to map the relationship between each class and individual in the ontology model to the Java classes that can be used in the system, mainly through constructors, properties, methods and so on.

4.2 Action Interaction

In the smart balcony scene, the main behaviors of users are to read the relevant data of washing machines and clothes drying hangers, start laundry, change the lifting state of clothes drying hangers, etc., which correspond to GET and UPDATE of action models.

GET Action. The interactive implementation of the GET action is divided into the following four steps:

Fig. 7. OWL Description.

- The client will initiate a GET request for the corresponding property to the server.
- The server decodes the communication data in a certain format, and parses it through the CoAP engine to obtain the result as shown in Fig. 8. The type is NON, indicating that the GET action does not require confirmation from the server, and the parameter in the URL is the corresponding property uri.
- Based on the URL and QUERY in the request, the server will determine whether there is any resource content that satisfies the conditions. If there is, the code will be put into the response payload, the response code will be set to 200, and a response will be sent to the client.
- After receiving the response, the client parses the response content and presents the payload information on the interface, as shown in Fig. 9. Actions and events appear selectively based on the property information. For example, "writeOnly" is true to make the UPDATE button available, and "observable" is true to appear related events.

```
D/../../messaging/coap/engine.c <coap_receive:185>:   type: NON
D/../../messaging/coap/engine.c <coap_receive:233>:   method: GET
D/../../messaging/coap/engine.c <coap_receive:245>:   URL: pro/mode
D/../../messaging/coap/engine.c <coap_receive:246>:   QUERY:
D/../../messaging/coap/engine.c <coap_receive:247>:   Payload:
```

Fig. 8. Parse data for GET request.

Fig. 9. The client interface of washing machine.

UPDATE Action. The interaction process of the UPDATE action is divided into the following five steps:

- After the user enters the target value of the property, the client sends an UPDATE request to the server.
- As can be seen from the Fig. 10, the type of the POST request is CON, indicating that this is a message that needs to be confirmed by the server. The payload of the request contains the desired property target value.
- The server will change the physical properties of the device according to the requirements, that is, modify the physical state of the actuation sensor.
- The server encodes the updated property value into the response payload, sets the response code to 200, indicating that the resource has been successfully changed, and returns this information to the client.
- After receiving the response, the client changes the value field in the property to the latest state.

```
D/../../messaging/coap/engine.c <coap_receive:182>:   type: CON
D/../../messaging/coap/engine.c <coap_receive:239>:   method: POST
D/../../messaging/coap/engine.c <coap_receive:245>:   URL: pro/mode
D/../../messaging/coap/engine.c <coap_receive:246>:   QUERY:
D/../../messaging/coap/engine.c <coap_receive:247>:   Payload: evalueiquickwash
```

Fig. 10. Parse data for UPDATE request.

4.3 Event Interaction

A major subclass of event is NOTIFY, which maps to the GET method on CoAP. The interaction flow of events is divided into the following steps:

– The client sends a request to the server to initiate a NOTIFY event.
– The trigger condition of the event is defined on the server side when the property is initialized. Once the NOTIFY event is started, the server will get the trigger condition of the event according to the property type. As shown in Fig. 11, when processing a request for a state property, the server first caches the property in the queue of the GET request, and then starts observing the state of this property.
– The type of the request is CON, so the server needs to return a message to confirm the confirmation to the client, put the uri of the observable property in the response payload, and return the code to the client.
– In the smart balcony scenario, the server will always observe the state property of the washing machine. The state value of this property is an object-type data, including four parameters: machine state, running time, remaining time, and progress percentage. During the laundry process, these four parameters will change regularly. Once changed, the server will record the changed state value.
– The changed property state value is placed in the response payload and sent to the client. The previous step and this step will continue to cycle until the laundry is completed.
– When the laundry is completed, that is, when the progress percentage is 100, the server will exit the above cycle, send a notification of this event to the client, and the corresponding client will receive a notification pop-up window.

```
D/../../messaging/coap/observe.c <coap_notify_observers:671>: coap_notify_observers: Issue GET request to resource /pro/state
D/../../messaging/coap/observe.c <coap_notify_observers:743>: coap_notify_observers: notifying observer
D/../../messaging/coap/observe.c <coap_notify_observers:813>: coap_observe_notify: forcing CON notification to check for client liveness
```

Fig. 11. The request for NOTIFY event.

4.4 Service Linkage

This section introduces the creation process of service linkage, including service creation of each device and multi-device invocation in the linkage model. After the creation is completed, the implementation process is described in detail.

The creation of service linkage is mainly divided into two parts: establishing the association between the service in the device and the required property and establishing the association between the service collection and the required service.

A schematic diagram of the washing machine creation service is shown in Fig. 12. After selecting the required property, the identifier of the service, the URI of the selected property and the device identifier will be stored in the service database, which will be also sent to the server for storage.

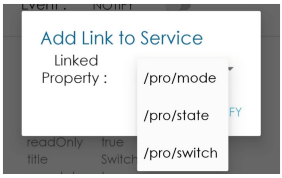

Fig. 12. The service creation of the washing machine.

After completing the storage of service information, the client will set relevant parameters in the service collection, read all service identifiers in the service database, to set relevant parameters for input trigger conditions and action trigger conditions, which will be sent to the server. Once the server operation is successful, the returned information is shown in Fig. 13.

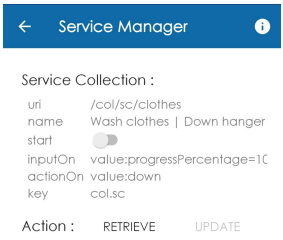

Fig. 13. The client interface of the service collection.

When the user sets the "start" parameter to "true", a POST request is sent to the server. After the service is started, the service manager will process the information in the service collection in turn, as shown in Fig. 14. This application scenario is about the linkage realization of washing machine and clothes drying hanger. The "state" property of the washing machine is used as the input, and the "updown" property of the clothes drying hanger is used as the action. According to the process, the service manager will always observe the value of the input. When it satisfies that the progress percentage is 100, the value of the "updown" property will be changed to the falling state, that is, the realization of "the laundry is completed, and the drying hanger is down".

5 Ontology-Based Model Evaluation

This evaluation of the ThingSpec model is carried out according to four parts: ontology reuse, language completeness, ontology extensibility and comprehensive assessment.

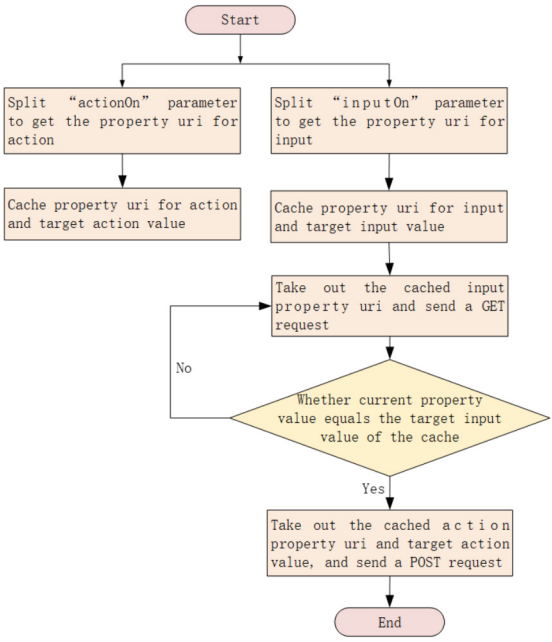

Fig. 14. Flow chart of server processing service collection.

5.1 Ontology Reuse

The ontology reuse refers to the probability that each URI is repeated in the ontology. To evaluate the ontology reuse, three conceptual variables and two computational variables are introduced as follows:

- Number of unique URIs used in the ontology N_{UN} [30]
- Number of URI name references used in the ontology N_N (i.e. every mention of a URI counts) [30]
- Number of URIs referencing other ontology sources N_{OSN}
- Ratio of name references to unique names $R_{NU} = \dfrac{N_{UN}}{N_N}$
- Ratio of URIs cited from other ontology sources to unique URIs $R_{OSU} = \dfrac{N_{OSN}}{N_{UN}}$

In order to further evaluate this performance, the above metrics are also applied to the SWoT ontology proposed by [21], which can be calculated according to the content of the ontology in Appendix B of the literature, as shown in Table 1. According to the results in the table, the metric R_{OSU} of the ThingSpec ontology is larger than that of the SWoT ontology, indicating that the ontology proposed in this paper has a higher proportion of the reusability of the existing ontologies; the R_{NU} of both meet the constraints, but the ThingSpec ontology has a lower ratio of name references to unique names, indicating that this ontology is better than the SWoT ontology in terms of URI reusability.

Table 1. The metrics of ontology reuse.

Metrics	ThingSpec Ontology	SWoT Ontology
N_{UN}	11	14
N_{OSN}	7	5
N_N	166	175
R_{OSU}	63.7%	35.7%
R_{NU}	6.63%	8%

5.2 Language Completeness

Language completeness measures the ratio between the knowledge that can be expressed and the knowledge that is stated [30]. If the two are equal and the ratio is 1, it means that the language completeness of the ontology has reached 100%, which is the most ideal state. The Resource Description Framework(RDF) is one of the languages used to describe ontology. For example, $ClassAssertion\,(C\ i)$ expresses that instance i is an entity object constructed based on class C. $PropertyAssertion\,(R\ i\ j)$ describes that instance i has an object property R, and the description object of R is the instance j. Then, taking RDF as an example, language completeness, that is, the completeness of defining "assertion", is based on the fact that all class assertions and property assertions contained in the ontology can confirm a description of whether a relationship exists between classes and instances or between instances.

Take the language fragment as "subject consistsOf object", that is, a form in which a certain class contains a certain class. The RDF description about this is as follows:

1. PropertyDomain(consistsOf Thing)
2. PropertyDomain(consistsOf ServiceCollection)
3. PropertyRange(consistsOf DataSchema)

The meaning of statement 1 and statement 2 is that the subject of the "consistsOf" can be a thing or a service collection. A thing class can be a device or a service manager. Statement 3 concludes that the object of the "consistsOf" should belong to the DataSchema, which includes actions, events, profiles, properties, services, and service collections.

Summarizing the above statements, it can be directly obtained that the value content of row 5 to row 11 (excluding the title row) in Table 2 is "×", that is, there is no inclusion relationship. The obvious relationship between classes can be obtained from Fig. 2, which are represented by "✓". There may be a relationship between the thing and the data shema, but there is no definite statement in this ontology, that is, it is represented by "?".

According to the results in Table 2, the number of the language fragments of "consistsOf" determined to exist and determined to not exist, that is, the number of "✓" or "×" in the table is 114, and the number of all sentences is 121, then $Ci = 114/121 = 94\%$, indicating that the part about the language fragment in this ontology model has good semantic integrity.

Table 2. Semantic statistics for "consistsOf".

consistsOf	1. Thing	2. Device	3. ServiceManager	4. DataSchema	5. Action	6. Event	7. Profile	8. Property	9. Service	10. ServiceCollection	11. Interaction
1. Thing	×	×	×	?	?	?	?	?	?	?	×
2. Device	×	×	×	✓	✓	✓	✓	✓	✓	×	×
3. ServiceManager	×	×	×	✓	×	×	×	×	×	✓	×
4. ServiceCollection	×	×	×	✓	×	×	×	×	✓	×	×
5. DataSchema	×	×	×	×	×	×	×	×	×	×	×
6. Action	×	×	×	×	×	×	×	×	×	×	×
7. Event	×	×	×	×	×	×	×	×	×	×	×
8. Profile	×	×	×	×	×	×	×	×	×	×	×
9. Property	×	×	×	×	×	×	×	×	×	×	×
10. Service	×	×	×	×	×	×	×	×	×	×	×
11. Interaction	×	×	×	×	×	×	×	×	×	×	×

5.3 Ontology Extensibility

The extensibility of ontology is based on the analysis of the possibility of extending classes on the basis of existing classes and properties.

For the data model, the current subclasses only contain basic data types, and can inherit more complex types of subclasses through inheritance, such as data in the database, file, or video. Correspondingly, profiles, actions, properties, events, services, and service collections based on the data model can also expand more complex triggering mechanisms and integrate more classes through links.

Due to the characteristics of ontology language, this model connects classes, instances, and data by adding object properties and data properties, which can be easily modified. The model itself has certain scalability. At the same time, each type of definition covers a wide range of application fields and is not fixed in a certain field. When used, subclasses and corresponding data can be added according to the particularity of the field, which has certain scalability.

5.4 Comprehensive Assessment of Model

To achieve the best practice of ontology, it is necessary to pay attention to: 1) reuse the existing ontology as much as possible to increase the reusability; 2) continuously adjust the ontology by reducing the heterogeneity between models and reducing the development time to make it extensible to improve interoperability [14].

The first point of the best practice is shown in Table 3. ThingSpec ontology reuses 4 existing ontologies, and the reuse rate of classes is relatively high, and the reused ontologies are widely used standard ontologies, which are more normative; the COIoT ontology [20] reuses 3 existing ontologies, showing multiple dimensions of the IoT, which are relatively comprehensive; the SWoT ontology [21] only reuses one ontology, and the reuse rate is low.

Table 3. Comparison of reuse cases.

Ontology	Reuse Ontology	Reuse Number
ThingSpec	WoT TD, SOSA, SAREF, Time	4
COIoT	SSN, PowerOnt, iot-lifecycle	3
SWoT	WoT TD	1

Regarding the second point of best practice, in terms of semantic description, all three have certain scalability. However, the COIoT ontology has no specific verification of application scenarios, so it cannot prove its effectiveness; although the SWoT ontology is verified by application examples on top of the complete description of the ontology model, based on only one ontology, it is mainly compatible with the semantic model of WoT, and its application scope is limited.

To sum up, combined with the two points of ontology best practice, the ThingSpec ontology not only occupies a large proportion in the reuse of existing ontologies, but also has certain scalability, and has the feasibility verification of application scenarios.

6 Conclusion

In this paper, based on multiple standard ontologies, the ThingSpec ontology model is designed, which combines the modules of thing model, static model, dynamic model and collection model, and reserves extension interfaces to solve the problem of semantic interoperability. At the same time, the model is applied to the OCF platform and implemented by taking a specific scenario as an example to verify the feasibility of the model. The related classes and instances constructed by the ontology software, combined with the metrics of the ontology, including the ontology reuse, the language completeness and the ontology extensibility. The evaluation shows that this model can not only realize the integration of multiple standard ontologies, but also has a high degree of the language completeness and extensibility.

However, this model still has some shortcomings and can be improved. For example, it can continue to supplement the data types in the data model, such as file data, audio and video data, and so on. In addition, the model proposed in this paper is only implemented on Android mobile phones to simulate scenarios. In the follow-up work, this model will be put into various physical devices for testing to prove that this model can be widely used possibility.

Acknowledgments. The authors would like to extend our sincere gratitude to the Open Connectivity Foundation (OCF) for their invaluable support. The OCF standard proposed by this organization has provided us with a robust open-source framework that has been instrumental in the implementation of the model discussed in this article.

We would also like to express our heartfelt thanks to all the dedicated organizations and researchers who have contributed to the development of the ontology model. Their collective efforts have laid a rich theoretical foundation for our study, and we are deeply appreciative of their contributions.

Our research journey has been enriched by the collaboration and assistance of many, and we are truly thankful for the collective effort that has made this study possible.

References

1. Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., Ayyash, M.: Internet of things: a survey on enabling technologies, protocols, and applications. IEEE Commun. Surv. Tutor. **17**(4), 2347–2376 (2015). https://doi.org/10.1109/COMST.2015.2444095
2. Noura, M., Atiquzzaman, M., Gaedke, M.: Interoperability in internet of things infrastructure: classification, challenges, and future work. In: Lin, Y.-B., Deng, D.-J., You, I., Lin, C.-C. (eds.) IoTaaS 2017. LNICST, vol. 246, pp. 11–18. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00410-1_2
3. Bröring, A., et al.: Enabling IoT ecosystems through platform interoperability. IEEE Softw. **34**(1), 54–61 (2017). https://doi.org/10.1109/MS.2017.2
4. Bröring, A., et al.: The big IoT API - semantically enabling IoT interoperability. IEEE Pervasive Comput. **17**(4), 41–51 (2018). https://doi.org/10.1109/MPRV.2018.2873566
5. Derhamy, H., Eliasson, J., Delsing, J.: IoT interoperability–on-demand and low latency transparent multiprotocol translator. IEEE Internet Things J. **4**(5), 1754–1763 (2017). https://doi.org/10.1109/JIOT.2017.2697718
6. Yang, S., Wei, R.: Tabdoc approach: an information fusion method to implement semantic interoperability between IoT devices and users. IEEE Internet Things J. **6**(2), 1972–1986 (2019). https://doi.org/10.1109/JIOT.2018.2871274
7. Tolcha, Y., et al.: Towards interoperability of entity-based and event-based IoT platforms: the case of NGSI and EPCIS standards. IEEE Access **9**, 49868–49880 (2021). https://doi.org/10.1109/ACCESS.2021.3069194
8. Mazayev, A., Martins, J.A., Correia, N.: Interoperability in IoT through the semantic profiling of objects. IEEE Access **6**, 19379–19385 (2018). https://doi.org/10.1109/ACCESS.2017.2763425
9. Adesina, T., Osasona, O.: A novel cognitive IoT gateway framework: Towards a holistic approach to IoT interoperability. In: 2019 IEEE 5th World Forum on Internet of Things (WF-IoT), pp. 53–58 (2019). https://doi.org/10.1109/WF-IoT.2019.8767248
10. Hatzivasilis, G., et al.: The interoperability of things: interoperable solutions as an enabler for IoT and web 3.0. In: 2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), pp. 1–7 (2018). https://doi.org/10.1109/CAMAD.2018.8514952
11. Mandal, A.K., Cortesi, A., Sarkar, A., Chaki, N.: Things as a service: service model for IoT. In: 2019 IEEE 17th International Conference on Industrial Informatics (INDIN), vol. 1, pp. 1364–1369 (2019). https://doi.org/10.1109/INDIN41052.2019.8972241
12. Novo, O., Francesco, M.D.: Semantic interoperability in the IoT: extending the web of things architecture. ACM Trans. Internet Things **1**(1) (2020). https://doi.org/10.1145/3375838
13. Gyrard, A., Datta, S.K., Bonnet, C.: A survey and analysis of ontology-based software tools for semantic interoperability in IoT and wot landscapes. In: 2018 IEEE 4th World Forum on Internet of Things (WF-IoT), pp. 86–91 (2018). https://doi.org/10.1109/WF-IoT.2018.8355091
14. Gyrard, A., Zimmermann, A., Sheth, A.: Building IoT-based applications for smart cities: how can ontology catalogs help? IEEE Internet Things J. **5**(5), 3978–3990 (2018). https://doi.org/10.1109/JIOT.2018.2854278

15. Mohammed, M., Romli, A., Mohamed, R.: Existing semantic ontology and its challenges for enhancing interoperability in IoT environment. In: 2021 International Conference on Software Engineering Computer Systems and 4th International Conference on Computational Science and Information Management (ICSECS-ICOCSIM), pp. 22–26 (2021). https://doi.org/10.1109/ICSECS52883.2021.00011
16. Haller, A., Janowicz, K., Cox, S., Phuoc, D.L., Taylor, K., Lefrançois, M.: Semantic Sensor Network Ontology. https://www.w3.org/TR/vocab-ssn/
17. TNO: SAREF: the Smart Appliances REFerence ontology. https://ontology.tno.nl/saref/
18. Simon Cox, C.L.: Time Ontology in OWL. https://www.w3.org/TR/owl-time/
19. Kaebisch, S., Kamiya, T., McCool, M., Charpenay, V., Kovatsch, M.: Web of Things (WoT) Thing Description. https://www.w3.org/TR/2020/REC-wot-thing-description-20200409/
20. Tayur, V.M., Suchithra, R.: A comprehensive ontology for internet of things (COIoT). In: 2019 Second International Conference on Advanced Computational and Communication Paradigms (ICACCP), pp. 1–6 (2019). https://doi.org/10.1109/ICACCP.2019.8882936
21. Antoniazzi, F., Viola, F.: Building the semantic web of things through a dynamic ontology. IEEE Internet Things J. **6**(6), 10560–10579 (2019). https://doi.org/10.1109/JIOT.2019.2939882
22. Open Connectivity Foundation: OCF Core Specification
23. Swamy, S.N., Kota, S.R.: An empirical study on system level aspects of internet of things (IoT). IEEE Access **8**, 188082–188134 (2020). https://doi.org/10.1109/ACCESS.2020.3029847
24. Yassein, M.B., Shatnawi, M.Q., Al-zoubi, D.: Application layer protocols for the internet of things: a survey. In: 2016 International Conference on Engineering MIS (ICEMIS), pp. 1–4 (2016). https://doi.org/10.1109/ICEMIS.2016.7745303
25. Phung, C.V., Dizdarevic, J., Jukan, A.: An experimental study of network coded rest http in dynamic IoT systems. In: ICC 2020 - 2020 IEEE International Conference on Communications (ICC), pp. 1–6 (2020). https://doi.org/10.1109/ICC40277.2020.9149026
26. Cheng, B., Zhao, S., Qian, J., Zhai, Z., Chen, J.: Lightweight service mashup middleware with rest style architecture for IoT applications. IEEE Trans. Netw. Serv. Manage. **15**(3), 1063–1075 (2018). https://doi.org/10.1109/TNSM.2018.2827933
27. Gao, W., Nguyen, J.H., Yu, W., Lu, C., Ku, D.T., Hatcher, W.G.: Toward emulation-based performance assessment of constrained application protocol in dynamic networks. IEEE Internet Things J. **4**(5), 1597–1610 (2017). https://doi.org/10.1109/JIOT.2017.2717386
28. Castellani, A.P., Loreto, S., Rahman, A., Fossati, T., Dijk, E.: Guidelines for mapping implementations: HTTP to the constrained application protocol (CoAP). https://rfc-editor.org/rfc/rfc8075.txt
29. Internet of Things (IoT)—Vocabulary
30. Vrandečić, D.: Ontology evaluation. In: Staab, S., Studer, R. (eds.) Handbook on Ontologies. IHIS, pp. 293–313. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-540-92673-3_13