From Software Services to IoT Services: The Modeling Perspective

I-Ling Yen[1], Farokh Bastani[1], San-Yih Hwang[2], Wei Zhu[1], Guang Zhou[1]

[1] University of Texas at Dallas
800 W. Campbell Road, Richardson, TX 75080, USA
ilyen@utdallas.edu

[2] National Sun Yat-sen University
70 Lienhai Rd., Kaohsiung 80424, Taiwan, ROC
syhwang@mis.nsysu.edu.tw

Abstract. Service ontology models have been applied to many application domains to facilitate the semantic rich specifications of various types of services, including the business processes, health care, manufacturing processes, etc. Recently, many service models for the Internet of Things (IoT) domain have been proposed. However, these models are still mostly following the thoughts of software services. In this paper, we discuss some differences of the IoT services from the software services and the requirements in service modeling for IoT services due to these differences. We also extend the existing software service model to support the specification of the IoT services and things.

Keywords. Internet of Things, IoT service model, service computing, service ontology.

1 Introduction

In recent years, Internet of Things (IoT) have gained increasing attentions in research community and industry. It is estimated that there are tens of billions of "physical things" that are connected to the Internet, and the number is still growing rapidly. Various IoT applications are continuously being developed towards the goal of more advanced automation and improved human living.

Many existing IoT systems are statically built. In these systems, the specific IoT devices and control and management software are statically selected and configured at the design time to achieve some predefined tasks and to handle some anticipated events. This type of systems has a similar nature as the conventional embedded systems, except that the constituent components (devices and software) are distributed in a wider area.

The IoT world interconnects a vast variety of capabilities, which can be so powerful if they are properly made use of, in addition to their statically assigned tasks. Consider a dynamic composition example. The police office receives a report of a hit-and-run incident that occurred 10 minutes ago by a red sedan at a location with coordinate

© Springer International Publishing AG 2017

Y. Hara and D. Karagiannis (Eds.): ICServ 2017, LNCS 10371, pp. 215–223, 2017.
DOI: 10.1007/978-3-319-61240-9_20

(x, y). The first task is to collect event related information, so police office will request the recorded videos, if available, from cars that may have been at location (x, y) 10 minutes ago. Then, another task will be to locate the culprit and request the image sensors on cars and on smart roads located within 10 minutes driving distance from (x, y) to detect the culprit. Both of these tasks are dynamic and the set of IoT devices cannot be identified in advance.

Service computing technologies can be leveraged to help with dynamic discovery and composition of the IoT devices to handle dynamically arising tasks. The fundamental technique for service discovery, selection, and composition is the service model. Without proper service specifications, the discovery and composition tasks will not work properly. However, existing service models are mainly designed for software services and may fall short for the modeling of IoT services. Some research works focus on the modeling issues for IoT services, such as encapsulating device control and interaction details and providing high level service interfaces [1] [2] and event based service modeling to manage the interactions among IoT services in an application system [3] [4], etc. Though these models are essential to IoT services, they mostly follow the same thoughts as software services, and do not consider some specific issues that are different in IoT services as compared to regular software services.

In this paper, we consider the insufficiency of existing service models for the IoT domain and discuss what should be considered in the IoT service model that have not been considered important or have not been considered at all in software service models. We then propose the modeling techniques to bridge the gap. The major issues we have identified and the modeling solutions are discussed in Sections 2 to 5. Section 6 surveys the existing IoT service models and discusses their potential problems. In Section 7, we conclude the paper and state potential future research directions.

2 Explicit Model for Things

In existing service models, the specifications of the services focus on their functionalities, not on the devices that can host the services. Also, a lot of research considers Quality of Service (QoS) issues during service composition and these works can address the availability, efficiency and other QoS issues of the service provisioning. But none of these need the specification of the underlying computing facilities that hosts the services. This is because software services are hosted by computing facilities that have sufficient uniformity and can be left out from the picture. On the other hand, the services provided by the IoT devices are mostly device specific and the characteristics of the devices can impact the service it provides. For example, all vehicles can provide the transport service, but each of them has its own characteristics, such as the capacity limit. If the cargo to be transported exceeds the limit of one truck, it is possible to select multiple trucks (different Things) or to select one truck and let it transport in multiple trips. Similar to software services, these issues may be left to QoS considerations. But due to the diversity of the devices, whether the devices are considered at the functional composition time or QoS based composition time, the specification of the IoT devices

should be explicit. Thus, besides specifying the services, the IoT service model requires the specific specification of the characteristics of the IoT devices.

There have been specification models proposed to specify devices that may be suitable for IoT systems. For example, OASIS has published the Devices Profile for Web Services (DPWS) [3] which defines the schema for specifying a device and its services. DPWS defines web Service description, discovery, messaging, and eventing for the device. However, there are two problems with this type of models.

(1) This model is device centric and services are defined as a part of the device specification. But frequently, the same service can be provided by a variety of devices. In this case, should we repeatedly provide the same service specifications for each device specification? We can also consider a service centric approach and for each service, define the devices that can provide the service. But this will raise the same issue. A device may be able to provide multiple different services, and the device specification should be repeated for the services.

(2) The specification for devices in DPWS is far from comprehensive. The major fields defined in DPWS schema are the device name, model, maker, etc. The essential properties of the devices are missing. For example, for a car, it is better to know its number of seats so that proper device allocation and scheduling can be performed.

Based on the observations above, we believe the IoT service model requires both the Services and the Things to be incorporated at the same upper level. They can be associated to each other in the upper ontology, instead of having one belong to the other. Also, the detailed specification for the Things should include QoS related properties that may impact the composition decisions. However, different set of attributes are required for different types of Things. Due to the diversity of Things, it is difficult to have a comprehensive model. Thus, domain specific ontology is needed to enhance the specification of the properties and profiles of Things. Fig. 1 shows the upper ontology for the IoT Service-Thing model (ST-model). For time being, the Service class can use the popular service models such as OWL-S, WSMO, etc. Later we will discuss the necessary extensions based on OWL-S for IoT service specifications. The expanded model for Things is shown in Fig. 2.

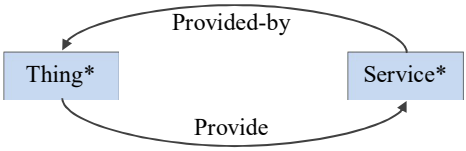
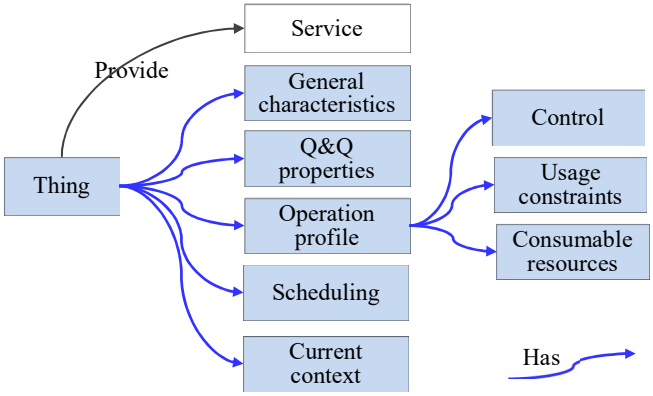


Fig. 1. Upper Service-Thing ontology for IoT.

In the Thing ontology, we try to incorporate the general classes for the specification of Things and leave the details to domain specific ontologies. The General characteristics class is similar to the definition given in the “Characteristics” class of DPWS. The Q&Q properties class is to specify the quantitative and qualitative properties that may impact the service selection decisions. For example, if we need to transport a group of

10 people from one location to another, it is important to know how many seats (quantitative property) are there in each car (Thing) in order to make the correct decision on service and thing selection for handling the transport task.

The Operation profile specifies the attributes that are related to how the Thing should operate. One important element in the Operation profile is the Control model, including the control mechanisms and commands for the Thing. Similar to the encapsulation feature in existing IoT middleware, the Control class can specify the detailed control commands and how the upper level services are mapped to a control mechanism, i.e., the



sequence of control commands. Also, Things may be nested. For example, a robotic swarm consists of multiple robots, which are also Things. In this case, the Control class can specify the mechanisms for the coordination of the lower level Things to achieve a certain service of the higher level Thing. These control mechanisms can also be specified in the Control class.

Fig. 2. Ontology for the Thing.

During service provisioning, there may be constraints on the provider Thing regarding how its services can be provided. For example, in most cases, multiple services provided by one Thing will have to be provided exclusively. Some device may have to be operated at a certain temperature range. These and other constraints can be specified in the Usage constraints class in the Operation profile of the Thing.

During the execution of software services, the computing facility would consume power. There are research works that consider how to minimize the energy consumption for services. Similarly, the operation of a Thing may consume some resources. A temperature sensor consumes battery power when providing its temperature sensing service. A truck consumes gas when providing its “cargo transport” service. Also, some Things requiring maintenances can also be viewed as requiring some consumable resources, e.g., a car would consume its maintenance free period. However, the issues of consumable resources for Things are different from the issues of energy consumption in software services. The energy consumption for software services can be handled as a QoS issue, while insufficiency of the consumable resources for the Things may require some external services to replenish them, which is a functional issue. Thus, the

resources and the sufficiency of the resources need to be exposed specifically in the specification of the Things. The event model is most suitable for this specification. The Consumable resources class can specify the resources needed and the events for insufficient resources. When such an event is triggered, external services can be activated to execute the replenishing task.

Generally, a software service has an execution context, but it hardly has much importance. In the physical world, the context of the Thing is very critical in service provision. For example, we cannot select a Thing in San Diego to fulfill a service required in Boston within an hour. Thus, the current context of the Thing and the context of the service request should be clearly specified. In fact, there is another important consideration that is not there for software services. Consider that a service consumer requests for a service at location X within a time limit T. A Thing t at location Y can provide this service. Then, we cannot just select t for the task. We also need to compose the transport services to bring t from Y to X within time T in order for t to properly fulfill the request. Here, we define the Current context class to specify the current context of a Thing. Later we will further discuss the issue of contexts in service composition.

A software service can be provided simultaneously to multiple requesters from different geographical locations, while IoT services may have to be provided with a specific context given in a request. Thus, scheduling has a significant role in the Thing-ontology. We define the Scheduling class in the Thing ontology to address the scheduling issues. For example, a plumber (Thing) provides a plumbing service. Several houses may require the plumbing service concurrently. The provider can only offer the service one at a time, and needs to schedule these requests and needs to request transport services to bring itself to these locations. A requester can choose to use another Thing in case one cannot provide a satisfactory schedule. Though scheduling can be considered as a QoS issue, it may trigger functional compositions due to the context issue.

3 Extending the IoT Service Model for the Contexts

We consider the OWL-S model for IoT services, but some extensions are needed to allow the service model to fully support IoT systems. We have already discussed the Apply-to extension in the IoT service model in Section 3. Here we consider the context requirements for the IoT services.

In the OWL-S model, a service is formally specified by its IOPE (inputs, outputs, preconditions, effects), where preconditions are the conditions that have to be satisfied before the service can be invoked and effects are the conditions that will hold after the execution of the service, if the preconditions are satisfied. A service request can be specified as an abstract service with its IOPE being the requirements for match making.

Here, we define "Context preconditions" to support the specification of the context requirements in a service request. Why can't the Context preconditions be specified as the regular preconditions? Generally, preconditions of a service are fixed conditions that stay the same for all service invocations. But Context preconditions are dynamic, and can probably be different in each invocation. Why can't the Context preconditions be specified as an input? The composition reasoning process needs to take the Context

preconditions into account, but input values are not considered during composition reasoning. Corresponding to Context preconditions, we also define the "Context effects" class to specify the dynamic effects that impact the states of the service recipients. The extended service model for IoT services is shown in Fig. 3.

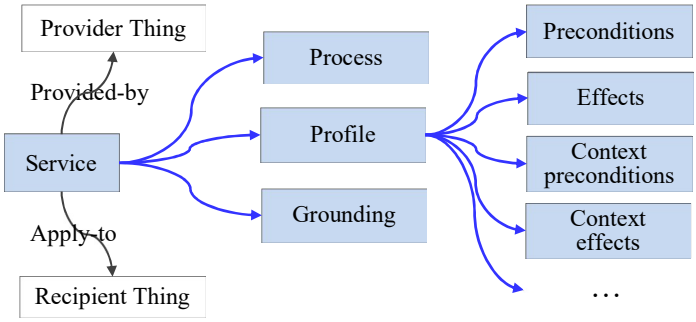


Fig. 3. Extended IoT Service model.

Separation of the regular preconditions/effects and Context preconditions/effects can also benefit staged composition reasoning. For example, consider a disaster site that is hard to reach by human rescuers. To reach the survivor-search goal, the functional reasoner selects a survivor-search service provided by a swarm of robots equipped with life detectors. The service has a Context precondition requiring that the provider Thing (the swarm) should be at the disaster site. To satisfy the goal, a functional composition reasoning is used to get a transport service provided by a truck. The QoS based composition reasoner can then select the truck that is closest to the swarm to provide the transport service. For complex composition problems, such separation can help reduce the complexity of the composition process.

4 Issues in Composition of IoT Services

In a composite service model, various composition constructs need to be provided to allow the services being composed by different execution patterns. For example, OWL-S supports sequence, split, split-join, choice, any-order, iterate, etc. The composition constructs in most existing service models only consider control flow constructs, which are not sufficient for IoT services.

Consider the example of survivor search by a swarm of robots. The robots cannot offer the survivor search service by themselves. The swarm can provide an "explore" service, but the effect of the explore service is traversing a region without any outcome. The robots need to be equipped with life detectors in order to provide the survivor search service. On the other hand, a life-detector Thing can provide the life detection service, but only within a small range. It is necessary to attached the life detector to a robot or a vehicle to fulfill the survivor search service. This type of composition cannot be reasoned if we only consider conventional control flow based composition constructs. We need some composition construct like "Apply", which allows a service x to

apply its service effects to its recipient service y, i.e., y's new effects become the aggregated effects of x and its own effects. When the "explore" service is applied to the "life detection" service, the composite service can be "survivor search".

In the physical world, things and their services sometimes need to coordinate with each other to accomplish a certain goal. For example, when a robot and its attached life-detector perform the survivor search service, a survivor is discovered but with some heavy object fell on him. The robot also offers a transport service, which can lift and move away the object. But in this situation, the object is too heavy for a single robot to transport it. The robot can ask another robot nearby to collaboratively lift and move away the object. During this "composed" effort, the behavior of one robot impacts the other and they need to communicate, synchronize, physically interact, etc. to successfully achieve the goal. The composition between collaborative Things and their services can be complex and may have many different patterns. In a more complex collaboration, many Things may have to collaborate to achieve the desired goal and the composition of their services can be even more complex.

Instead of considering each case of collaboration, we consider both example cases given above in a uniform paradigm and provide a collaboration composition construct to support the specification of such compositions. The model for the collaboration composition construct is shown in Fig. 4.

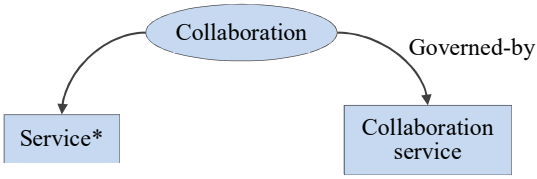


Fig. 4. Upper Service-Thing ontology for IoT.

The collaboration between the services is governed by a collaboration service, which provides messaging, synchronization, event management, as well physical actions required for the collaboration. As can be seen, the collaboration composition may have a little similarity with the conversation model, but is far more complex. It can be used to in place of the conversation model to offer conversational services. For the first example above, the collaboration service should specify that the robot should attach the life detector to itself and define the effects of the collaborative service based on the effects of the collaborative services. For the second example above, the collaboration service needs to provide the synchronization mechanism for the collaborative services and the collaborative services need to define all their synchronization points.

5 Brief Literature Review

There have been several service modeling research directions for IoT in the literature. One direction defines the event model, including event definition, publishing, and sub-

scription, to model IoT service interactions. DPWS [3], SOCRADES [1], SenaaS (sensor as a service) [2], and ED-SOA [4], are example works in this category. Though event model is indeed suitable for IoT services, the modeling concept and mechanism are almost the same as those for software services [5]. Another major direction is designing middleware to support the encapsulation of detailed control of IoT devices and offering web service interfaces to simplify the IoT capability invocation. Representative works include SOCRADES [1], SenaaS (sensor as a service) [2], and ScriptIoT [6] are example middleware for IoT. Encapsulation is essential for the diverse types of IoT devices, but the concepts and designs in these middleware are not much different from conventional middleware for software systems. Some IoT models focus on sensors and sensor data, such as SensorML [7] and SSN-Ontology [8]. These models are data driven, in which services are composed for sensor data processing or for reacting to events detected from the IoT data. None of these existing works can address the issues in service modeling and composition for IoT services that are different from those in software services.

6 Conclusion

In this paper, we take a different approach form other research works in IoT service modeling and look into the issues in IoT service modeling that are different from existing software services models. We then attempt to construct a more comprehensive IoT service model to address these issues.

We plan to consider a set of example cases to evaluate our IoT service model and further investigate the missing issues. Also, as discussed in some examples in the paper, the composition reasoning process for IoT services may be complex and it is better to decompose the composition process into multiple stages, including functional and QoS-based compositions. We are developing automated composition techniques for IoT services based on this decomposition approach.

References

1. P. K. S. Spiess, D. Guinard and D. Savio, "SOA-based integration of the Internet of Things in enterprise services," in International Conference on Web Services, 2009.
2. S. Alam, M. M. Chowdhury and J. Noll, "SenaaS: An event-driven sensor virtualization approach for Internet of Things cloud," in IEEE International Conference on Networked Embedded Systems for Enterprise Applications, 2010.
3. T. Nixon, "OASIS Devices Profile for Web Services (DPWS), Version 1.1," 2009. http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.pdf.
4. Y. Zhang, L. Duan and J. L. Chen, "Event-driven SOA for IoT services," in IEEE International Conference on Service Computing, 2014.
5. S. Graham, P. Niblett, D. Chappell, A. Lewis, N. Nagaratnam, J. Parikh, S. Patil and etc., "Publish-subscribe notification for Web services, Version 1,0," 2004. https://www.oasis-open.org/committees/download.php/6661/WSNpubsub-1-0.pdf.

6. H.-C. Hsieh, K.-D. Chang, L.-F. Wang, J.-L. Chen and H.-C. Chao, "ScriptIoT: A script framework for Internet-of-Things applications," IEEE Internet of Things Journal, pp. 628-636, 2015.
7. M. Botts and A. Robin, "OGC SensorML: Model and XML encoding standard," 2014. http://www.opengis.net/doc/IS/SensorML/2.0.
8. M. Compton, P. Barnaghi, L. Bermudez and e. al., "The SSN Ontology of the Semantic Sensor Networks Incubator Group," in Web Semantics: Science, Services and Agents on the World Wide Web, 2012.