Service-Oriented IoT Modeling and its Deviation from Software Services

I-Ling Yen, Farokh Bastani,
Wei Zhu, Hessam Moeini
University of Texas at Dallas
{ilyen, bastani, wxz094120, hxm141530
@utdallas.edu}@utdallas.edu

San-Yih Hwang
National Sun Yat-Sen University
syhwang@mis.nsysu.edu.tw

Yuqun Zhang
S. Univ. of Science and Technology
zhangyq@sustc.edu.cn

Abstract. Service technologies have been widely applied to many application domains to facilitate rapid system composition and deployment. However, existing service models need to be enhanced in order to be used in Internet-of-Things (IoT). Also, due to the massive-scale, IoT service discovery and composition cannot be centralized. Existing discovery routing protocols for peer-to-peer systems have their shortcomings and need to be improved. In this paper, we analyze the differences between IoT services and software services and identify the requirements for designing IoT service models that are additional to software service models. We then discuss a service ontology model for the specification of IoT services. For IoT service discovery, we survey existing discovery routing approaches, including those for conventional peer-to-peer networks and for IoT systems and discuss the potential problems when used in IoT networks. Then, we discuss our approach, summarization and ontology coding, which greatly reduce the memory requirements of the routing protocols, for the IoT networks.

Keywords. Internet-of-Things, service discovery and composition, service discovery routing, peer-to-peer.

1 Introduction

Service oriented architecture (SOA) has become the major architecture model in modern system development process. With the advances in various hardware and networking technologies, computing devices becomes more and more pervasive in human society and in our daily lives. Nowadays, Internet of Things (IoT) have gained increasing importance. Consequently, many researchers are investigating SOA technologies for IoT system development.

In the beginning of SOA research, the focus was on its use in the development of enterprise systems. Thus, the main research directions include the service-oriented architecture itself as well as technologies for service discovery and composition. With the centralized registry UDDI, there is no routing issue and, hence, how to model and specify the available services and the to-be-composed systems became the most important issue. Service modeling is the foundation for matchmaking and grounding. Widely used specification models such as WSDL, OWL, WSMO, etc. provide different levels of sophistication in service modeling.

With the increasing availability of software services cross organizations all over the globe, service computing has been globalized beyond enterprises. During this transition, service discovery has to take routing issues into consideration since it is no longer feasible for a central UDDI to tracking all available services. Various hierarchical search solutions and decentralized routing protocols have been investigated.

As we come to the IoT era, SOA technologies need to take another stage of evolution. This evolution should include both the modeling perspective as well as the service discovery routing protocols.

In terms of modeling, IoT services are quite different from software services and modeling for IoT services should be carefully investigated. Existing SOA models are developed with software services in mind and modifications are required to support better IoT service specification and matchmaking. However, current SOA models for IoT systems mostly follow the same design concepts as those considered in software systems. In IoT world, most of the devices have proprietary access commands and data retrieving protocols. Some SOA-based middleware systems encapsulate Things and wrap them to offer more uniform service invocation interfaces [1] [2] [3] [4]. Many IoT applications can be better designed with event-based modeling, and event-based SOA models have been proposed to fulfill the needs in these systems [1] [3] [5] [6] [7]. These middleware design and modeling techniques are very important for IoT systems, but there are no novel issues when applied to the IoT world.

In [8] [9], we have investigated the insufficiency of existing service models for the IoT domain and what should be considered in the IoT service models that have not been considered important or have not been considered at all in software service models. We have also proposed the modeling techniques to bridge the gaps, attempting to make IoT service models more complete and support better IoT service composition reasoning.

On the routing side, we consider routing in dynamic IoT systems. Many existing IoT systems are statically built and the specific IoT devices and control/management software are statically designed 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 Things are interacting via Internet instead of proprietary buses, power wires, etc. The major issues for communication among Things in these systems include communication protocol standardization and interoperation between various protocols and routing is an insignificant issue. However, the IoT world interconnects a vast variety of

0-7695-6394-5/18/$31.00 ©2018 IEEE
DOI 10.1109/SOSE.2018.00014

capabilities, which can be so powerful if they are properly made use of, in addition to their statically assigned tasks. Thus, it is inevitable to consider dynamic IoT systems, in which IoT services can be discovered and composed dynamically to handle dynamically arising tasks. In these dynamic IoT environment, routing becomes highly critical so that Things can be discovered and invoked in a timely manner in order to cope with the dynamically occurring situations.

Since the number of Things in the IoT world are massive and Things are widely distributed, thus, centralized solutions become infeasible and decentralized routing becomes an essence. There are many decentralized routing algorithms in the peer-to-peer literature. Among them, the DHT (distributed hash table) based solutions [10] are not suitable for IoT systems since IoT nodes cannot be arbitrarily moved to their hashed locations. The caching based selective flooding algorithms are suitable for large scale peer-to-peer networks, especially when mobile IoT nodes are considered [11] [12]. However, since they are designed for general computing systems, they do not consider the resource constraints on IoT devices. With limited memory space on many IoT devices, the cache size will be small and the system needs to throw away useful routing information, which hurts the discovery routing performance.

We consider the cache based approach for routing service discovery queries. To cope with the very limited cache size on IoT devices, we have developed a summarization technique to help retain the useful routing information in the cache and, hence, improve the routing performance [13] [14]. In our approach, instead of throwing away useful routing data, we summarize them to retain the potentially useful routing information while significantly reduce space requirements. Also, how to effectively summarize routing information can be challenging. We organize the capabilities of the IoT devices in an ontology tree, in which leaf nodes are the actual capabilities of the IoT devices and the internal tree nodes are the summarized capabilities. We then make use of the semantics in the ontology tree structure to achieve effective capability summarization.

In this paper, we survey existing solutions for modeling IoT services and discuss some routing protocols that have been considered for IoT service discovery (Section 2). Based on the literature, we investigate the issues remain to be investigated and discuss our solutions that can bridge some of the gaps. In Section 3, we discuss our IoT service model which addresses the deviation of the IoT services from software services. In Section 4, we discuss a summarization concept we have developed and the mechanisms that help realize the concept. Section 5 concludes the paper.

2 Related Work

2.1 Service Models for IoT Systems

In recent years, Internet of Things (IoT) and cyber physical systems (CPS) attracted intensive research. Since existing service models are mainly designed for software services and may not suit IoT/CPS services, some research works attempt to adapt the models for IoT/CPS systems. Here, we survey the works in IoT/CPS service modeling.

Service middleware focusing on encapsulation. In IoT, various Things usually have very different and complex access interfaces (involving low-level control sequences) and different communication protocols. Thus, it is very important to extend the wrapping and encapsulation techniques in software world to unify the access interfaces for the IoT services. Some IoT middleware systems are developed to encapsulate the devices and providing a uniform interface to access devices that offer similar functionalities.

The SenaaS [1] middleware consists of three layers: the service virtualization layer, the semantic layer and the real-world access layer. The real-world access layer provides unified interfaces for accessing similar services provided by functionally similar IoT devices. These devices with different access protocols and communication mechanisms are wrapped in this layer. The semantic layer provides the needed ontologies in the middleware to support the service specifications in different layers, including the sensor ontology, an event ontology, and the service access policies. These ontologies can facilitate cross layer mappings and enhance the effectiveness of device encapsulation. The virtualization layer provides users with "virtual Things" that may be the composition of several "real Things". Since SenaaS focuses on sensors, the middleware essentially offers virtual sensors whose output data are the aggregation of the data retrieved from multiple real sensors.

In ScriptIoT [2], a common script interface is provided to access IoT sensors with different data formats and communication mechanisms and to activate different IoT devices. For example, a common "fetch(d)" command can be used to fetch data from a sensor d of any type and the underlying accesses protocol for d and conversions of d's data format are encapsulated. Such sensor data accesses can also be registered as an event and only when the data satisfying a certain condition will the event be delivered to the request issuer.

In [3], the proposed middleware also consists of three tiers and it focuses not only on sensors but also on the control activities in response to sensor readings. The lowest layer is the environmental tier, which encapsulates the physical devices. The control tier consists of controllers. Each controller subscribes to specific monitoring data stream gathered by sensors, analyzes the sensor data to make control decisions, and composes services to realize the control decisions. The service tier analyzes common services needed by the control tier and composes the functionalities provided by the IoT devices to realize these identified common services. Thus, the specific accesses to the devices are not exposed to the control tier or to the users to avoid complex access procedures. In [4], a similar 3-tier architecture is considered and the goal is also to encapsulate the interaction protocols with the devices and conduct service composition to react to situations.

Event driven IoT/CPS service models. Similar to some software services, physical services may be invoked by requests and/or by events. For example, air conditioner may be turned on (invoked) due to a high temperature reading from the temperature sensor or by the user. When the gas tank of a car is low, the gas filling service should be invoked before providing the car transporting services. Some research works use the event-based model from soft*ware systems to capture these event driven characteristics in the IoT world.

In [5], an event-driven service-oriented architecture (ED-SOA) for IoT systems has been proposed. In this model, events are treated at the same level as services. A service can subscribe to a set of events and it takes corresponding actions when some of these events are delivered to it. Also, events may be generated during service executions and they will be delivered to their subscribers. DPWS (devices profile for Web services) [6] is a standard defined by OASIS (led by Microsoft) for resource constrained devices to interact securely. It also uses an event-based model following the same publish-subscribe model. DPWS also considers device access protocol specification and service specification and discovery for the IoT devices. The SenaaS (sensor as a service) system [1] is a sensor virtualization framework for IoT cloud. It also uses the event-driven SOA in its virtualization layer, which is in charge of receiving events, managing them, and sending them to subscribers so that appropriate actions can be taken. SOCRADES [7] explicitly provides an event system. The status information of the physical entities and sensors can be defined as events and can be subscribed by other services through the SOCRADES event system. Also, a virtual composition language is defined in SOCRADES to specify the bindings of events, event handling services, and the corresponding Things.

In [3], an event-based model is used as the underlying system model. But unlike the other systems discussed above in which the event handling logic is manually determined in advance, this work emphasizes to dynamically compose services to handle events. After an event is raised, the control layer determines a control decision for the event. Then the services are composed together to realize the control decision. Also, since some physical devices are configurable, the services incorporate configurability to support flexible provisioning. Though the dynamic control decision making and on-the-fly service composition are important for unexpected situations, there is no effective methods in this framework to support such goals.

Modeling the Things that provide the services. The physical Things in IoT systems have a significant role in the composition reasoning of physical services, which is very different from composition reasoning for software services. In software services, the Thing is the computing and storage hardware. However, due to the sufficient uniformity in the computing facilities for software services and the high speed communication among them, though there are still issues like communication costs and workloads, the Things for existing software services do not have a significant role. In IoT, the physical Thing that provides a service and its properties are very important. For example, different types of vehicles can be used to transport people from a disaster site to a safe evacuation area. But each type of vehicle has its own characteristics, such as load capacities and number of seats. Also, even for the vehicles that are exactly the same, when grounding the service for transporting people, it is necessary to specifically determine the number of vehicles required and the number of trips each vehicle may have to make. The second issue for physical service composition is that, a Thing may be able to provide several different types of services. However, it is frequently not possible for one Thing to fulfill multiple services it provides at the same time. The schedule of individual Thing will impact the service composition result. The Thing context is also an issue during physical services composition. The context is defined as the dynamic changing states of a Thing. For example, in a rescue mission, some robots may be used for survivor detection. The physical location of the robots must be at the rescue site. If not, additional services are required to bring them to the rescue site. Last issue that needs to be considered is that, the side effect of a software service generally can be specified independent of other software services. This may not be true in physical things. For example, a car may transport a robot to a disaster site for a rescue search. In this case, the states of the Thing that provides the service and the recipient Thing of the service may change together. Such impact need to be specified explicitly and existing software service models do not have such a feature.

In existing models, there are efforts toward the modeling of things. The lower level specifications in DPWS (Devices Profile for Web Services) [6] and EDDL (electronic device description language) [15] provide device specifications, but they focus on the interactions with the devices, not about the properties of the devices themselves. In SOCRADES [7], the availability of a device for service provisioning is considered as an event and the broker will deliver this type of events to the subscribers. This offers some help in service composition, but only on the availability of the devices, not on other properties and constraints of the Things.

Remarks. The event based modeling may be useful in IoT world, but it does not offer additional features compared to the event modeling for software services. The middleware for wrapping and encapsulation is a direct application of the concepts in software systems to IoT systems. Though they are important for IoT systems, they do not address some of the issues discussed in this section. As to the specification models for the physical Things, existing models are not designed with the IoT service technologies in mind. In general, they only consider how to interact with the IoT devices, not those attributes that are important when considering IoT service matchmaking and composition.

2.2 IoT Service Discovery Routing

Dynamic IoT service composition requires dynamic IoT service discovery. Many IoT routing protocols are IP based (e.g., RPL) or ID based (e.g., EPCglobal), which cannot help with functionality based IoT service discovery.

Existing semantic-based routing protocols can be applied to IoT networks by defining the capabilities of IoT devices using keywords and concepts and route according to the capabilities. This technique is commonly used in semantic based service discovery protocols. Some of the semantic based routing are centralized, which may not be able to scale for widely distributed IoT systems, especially when we consider mobile IoT devices, which may cause frequent updates and result in significant communication overhead.

Hierarchical routing protocols can improve on the scalability problem. In Rendezvous Regions [16] and Service Rings [17], services are divided into groups based on their proximity and semantic similarities. In Rendezvous Regions, some routing protocol (unspecified) can be used to route queries to the desired group. Within the group, a flooding based approach is used to locate the desired host. In Service Rings, hierarchical groups are formed based on the same principle (proximity and semantic similarities). Each group elects a leader to serve as a directory server. Routing is done hierarchically by traversing the service rings. These approaches are also called clustering based approaches. Cluster based approaches can also facilitate service selection based on QoS requirements. However, they still require inter-cluster routing and specific routing protocols are needed to complete the solutions. Some hierarchical networks are specifically designed for IoT [18] [19]. In [18], the global network is divided into many local networks, each with a gateway centrally managing the local IoT nodes. Gateways are then networked together using existing peer-to-peer network solutions. It is not clear how routing can be done efficiently in the upper layer peer-to-peer network. [19] has a similar architecture as that in [18], except that they consider federating autonomous local IoT networks.

Decentralized routing protocols include structured and unstructured solutions. Structured semantic routing solutions are generally DHT (distributed hash table) based, which hashes the resources or services to specific servers. DHT has been used in peer-to-peer service discovery [20] [21] [22] [10]. In [20] [22], services need to be deployed at their hashed locations, which has no problem for software services, but are not applicable to IoT services because the IoT devices cannot be flexibly moved to their hashed locations. [22] [10] also considers service description using multiple keys, which is very important for software services, while IoT capabilities generally can be specified more specifically without using multiple keywords. [10] specifically use DHT for IoT service discovery. The hashed destination nodes are used as indirect pointers to locate the actual services, which imposes additional communication overhead (double the cost). Generally speaking, the advantage of DHT only emerges in systems with a very large keyword space compared to the number of physical nodes, so that hashing provides a uniform distribution of the objects. In IoT networks, many IoT nodes may have the same capabilities and the number of different IoT capabilities may be similar or even less than the number of physical nodes. Thus, most of the nodes in the network will not contribute to routing since nothing will be hashed to them. Thus, DHT is in general not a good solution for IoT.

Unstructured decentralized routing protocols are mostly flooding based. The older routing protocol, Gnutella [23], uses pure flooding. It does not require any memory space for storing the routing table, but the service discovery phase incurs heavy network traffic due to the flooding nature. When there are multiple service discovery queries, network may get congested and cause delays. The caching approach [11] [12] is based on selective flooding. A node upon receiving a query, does not forward it to all its neighbors, but selects the most promising ones for forwarding. The intermediate nodes on the discovered path caches the routing information and use it for future flooding path selection in order reduce the routing costs [20] [24]. These schemes may result in a large cache size, potentially having one entry for every capability in the system. To control the cache size, GSD [12] adds a hop limit and incorporates the service group concept. A node needs to cache the service name and all the names of the corresponding service groups for each resource in its neighborhood. This design results in inefficient use of memory space and large messages size, making it not suitable for IoT systems.

Bloom filter (BF) has been used in semantic based routing [25] [26]. Generally, BF are used in the supernode structures where supernodes exchange the keywords (which is mapped to BF) with their peers. BF is very useful when the number of keywords to be passed is reasonably high relative to the entire keyword space. In an IoT network, the capabilities of each node can generally be described by a few keywords. But the number of keywords for describing all the capabilities in the network can be relatively high. Thus, using keywords directly can be much more space efficient than that of BF solutions.

Remarks. The centralized and hierarchical IoT service discovery routing solutions incurs communication overhead for registry updates, especially if the IoT network is dynamic due to mobility or other factors. DHT solutions is not suitable for IoT systems. Caching based peer-to-peer solutions have potential advantages for IoT service discovery. However, the cache size limits on resource-constrained IoT nodes could force the cache to throw away useful information, resulting in ineffective discovery routing.

3 From Software Services to IoT Services

3.1 Modeling the 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 (e.g., DPWS [6] and EDDL [15]) 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.

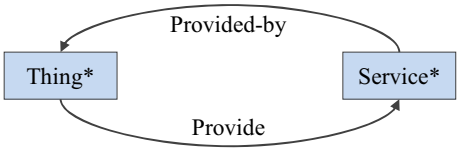

Figure 1. Upper Service-Thing ontology for IoT.

From the above, we believe that 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. Figure 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 Figure 2.

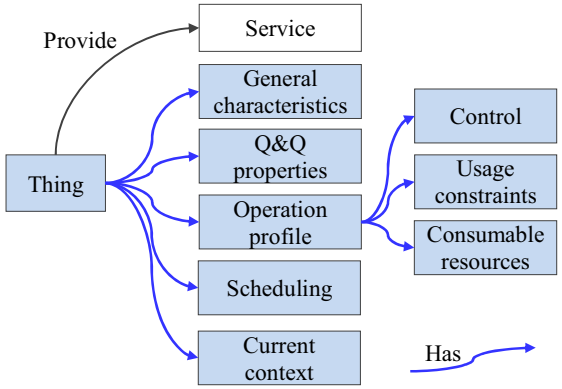

Figure 2. Ontology for the Thing.

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 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.

During service provisioning, there may be constraints on the provider Thing regarding how its services can be provided. For example, 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. 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.

3.2 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 Figure 3.

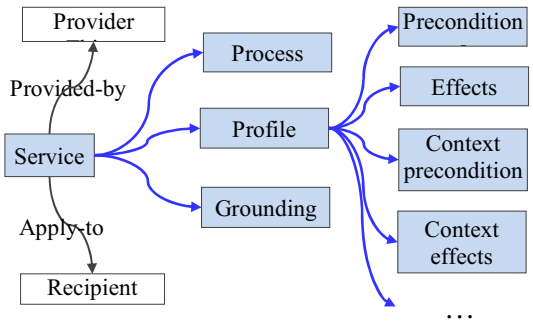

Figure 3. Extended IoT Service model.

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 Routing for IoT Service Discovery

Many IoT devices have limited resources, especially

memory. Table driven and cache based routing protocols require a table (or cache) to store useful routing information so that service discovery queries can be routed efficiently without incur a high network traffic volume. Most of the existing semantic based routing algorithms do not consider bounded table (cache) size since they are designed for general computing systems. A reasonable resolution is to delete some routing data when the routing table size exceeds a given limit. But by doing so, some potentially useful routing information may be lost. In our approach, instead of throwing away routing data, we summarize them to retain the potentially useful routing information while significantly reduce their space requirements. But, how to effectively summarize routing information can be challenging. In our solution, we organize the capabilities of the IoT devices in an ontology tree to facilitate ontology-based summarization. An example ontology is shown in Figure 5.

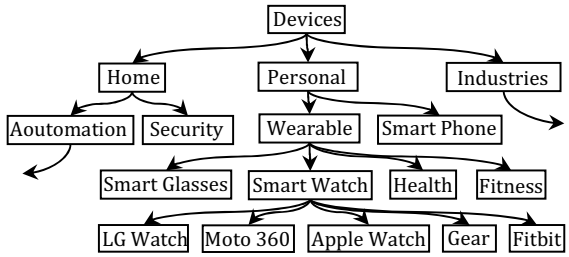

Figure 5: A sample Ontology

In the ontology tree, leaf nodes are the actual capabilities of the IoT devices and the internal tree nodes are the summary IoT capabilities of different degree. For example, the system may summarize several different watches in the neighborhood into the "SmartWatch". However, in order to perform summarization at each IoT node, it is necessary for the IoT node to store the ontology and the size of the ontology tree could be quite significant. This fully defeats our purpose of summarization and attempting to have the routing operation use a very small memory size. To make the summarization concept workable, we design an ontology coding scheme to code the ontology. For each IoT capability, only a small ontology code need to be used and at the same time, this small code carries sufficient information for the IoT node to perform summarization without needing to store the entire ontology tree.

Here we define the ontology code. Let $ONID(on)$ denote the ontology id of an ontology node on. $ONID(on)$ consists of two bit vectors, including the "ID" bit vector and the "SP" bit vector. The ID vector specifies a code for each ontology node. It is an aggregation of codes level by level from root to the node in the ontology tree. The SP vector specifies the "starting position" of each level of code. Let $ONID(on).ID$ and $ONID(on).SP$ denote the ID and SP bit vectors of an ontology node on. $ONID(on).ID$ includes the parent code, $ONID(on).PID$, and the sibling code, $ONID(on).SibID$. $ONID(on).PID$ is essentially the ID vector of *on*'s parent. The sibling code is a unique code among the siblings of node *n*. More formally, we have

$$ONID(on).ID = ONID(on).PID \bullet ONID(on).SibID$$

$$ONID(on).PID = ONID\big(parent(on)\big).ID$$

$$ONID(on).SibID = pos(n, chlist(parent(on)))$$

Here, function $parent(on)$ returns the parent node of *on* in the ontology, $chlist(on)$ returns the list of child nodes of node *on*, and $pos(on, l)$ returns the position of *on* in list l (assume that *on* is an element of *l*). To uniquely define the sibling code $ONID(on).SibID$, its code length should be

$$\|ONID(on).SibID\| = \lceil \log_2 \|chlist(parent(on))\| \rceil$$

Figure 6 shows an example of coded ontology nodes. The root ontology node has an assigned code "0". The root has five children. For all the child nodes $i, 1 \le i \le 5$, their $ONID(i).PID$ should be "0", $\|ONID(i).SibID\|$ should be 3 bits, and $ONID(i).SibID$ should be "000", "001", "010", "011" and "100". As shown in the figure, the same coding scheme is applied to the three child nodes of "0001".

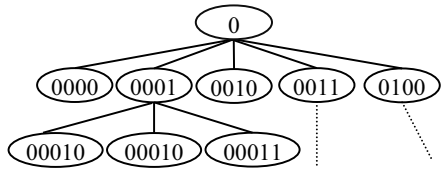

Figure 6. $ONID(on).ID$

As can be seen, the coding scheme defined above will result in different code length for each ontology node. If we simply pad the code, then the ID for each node may not be unique. More importantly, from $ONID(on).ID$, we cannot decompose the code to recognize $ONID(i).PID$ and $ONID(i).SibID$ which is essential for identifying the relations between nodes.

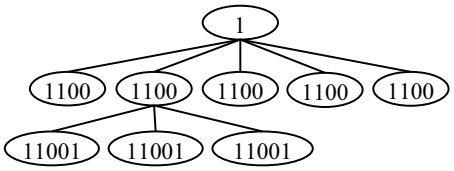

Figure 7: $ONID(on).SP$ vector coding

We use an SP vector, $ONID(on).SP$, to solve the problem. $ONID(on).SP$ specifies the "starting position" of each level of code in $ONID(on).ID$ (by setting the bit to 1 at the start position). Similar to $ONID(on).ID$, $ONID(on).SP$ includes the parent SP, $ONID(n).PSP$ and the sibling SP $ONID(on).SibSP$. Formally, we have

$$ONID(on).SP = ONID(on).PSP \bullet ONID(on).SibSP$$

$$ONID(on).PSP = ONID\big(parent(on)\big).SP$$

$$\|ONID(on).SibSP\| = \lceil \log_2 \|chlist(parent(on))\| \rceil$$

$$ONID(on).SibSP[i] = \begin{cases} 0, \text{if } i = 0 \\ 1, \text{otherwise} \end{cases}$$

Figure 7 shows the SP vectors for the sample ontology code given in Figure 6.

Now we can pad $ONID(on).ID$ and $ONID(on).SP$ with 0's for all ontology nodes to the same length while retaining the uniqueness of ONID for all nodes. The ONID for each node is the concatenation of $ONID(on).ID$ and $ONID(on).SP$ and the code length of $ONID(on)$ is the maximal code length of any node in the ontology as derived earlier. Based on the ontology code, the IoT system can easily achieve the summarization goal in the routing table (or cache) without storing any part of the ontology tree. Note that summarization can only be performed for the routing table entries under the same neighbor.

5 Conclusion

IoT service discovery and composition research requires evolutions in two areas: IoT service modeling and IoT service discovery routing. We have surveyed existing service modeling literature and discussed how the considerations for IoT services are different from software services from the modeling perspective. We then extend OWL-S and other service models to build the IoT-specific service model. For discovery routing, we have surveyed existing solutions and identified that the caching based protocols are most suitable for IoT networks. Since existing cache based protocols do not consider cache size bound due to the limited memory space on IoT devices, we discuss our solution, the summarization and ontology coding, and how it can be used to bound the memory requirement without significantly degrade the routing performance.

6 Reference

[1] 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 .

[2] H.-C. Hsieh, C. Kai-Di, W. Ling-Feng, C. Jiann-Liang and C. Han-Chieh, "ScriptIoT: A Script Framework for Internet-of-Things Applications.," IEEE Internet of Things Journal, pp. 628 - 636, 2015.

[3] H. J. La and S. D. Kim, "A service-based approach to designing cyber physical systems," in 2010 IEEE/ACIS 9th International Conference In Computer and Information Science (ICIS), 2010.

[4] Z. Song, A. A. Cárdenas and R. Masuoka, "Semantic middleware for the internet of things," in Internet of Things , 2010.

[5] Y. Zhang, L. Duan and J. L. Chen, "Event-driven soa for iot services," in IEEE International Conference on Service Computing, 2014.

[6] T. Nixon, "OASIS Devices Profile for Web Services (DPWS) Version 1.1," 2009. [Online].

[7] A. Cannata, M. Gerosa and M. Taisch, "SOCRADES: A framework for developing intelligent systems in manufacturing," in IEEE International Conference on Industrial Engineering and Engineering Management , 2008.

[8] W. Zhu, G. Zhou, I.-L. Yen and F. B. Bastani, "A PT-SOA model for CPS/IoT services," in ICWS, New York, July 2015.

[9] I.-L. Yen, F. B. Bastani, S.-Y. Hwang, W. Zhu and G. Zhou, "From software services to IoT services: The modeling perspective," in International Conference on Serviceology (ICServ), July 2017.

[10] F. Paganelli and D. Parlanti, "A DHT-based discovery service for the Internet of Things," in JCNC, 2012.

[11] O. Ratsimor, D. Chakraborty, A. Joshi and T. Finin, "Allia: Alliance-based service discovery for ad hoc environments," in Mobile Commerce Workshop, Sep. 2002.

[12] D. Chakraborty, A. Joshi, Y. Yesha and T. Finin, "GSD: A novel group-based service discovery protocol for MANETS," in IWWCN, 2002.

[13] H. Moeini, I.-L. Yen and F. B. Bastani, "Routing in IoT networks for dynamic service discovery," in IEEE International Conference on Parallel and Distributed Systems (ICPADS), , Shenzhen, China, Dec. 2017.

[14] H. Moeini, I.-L. Yen and F. B. Bastani, "Efficient caching for peer-to-peer service discovery in Internet of Things," in ICWS, 2017.

[15] J. Berge, "Electronic device description language (EDDL) for efficiency," HydroCarbon Asia, pp. 46-54, March-April 2008.

[16] K. Seada and A. Helmy, "Rendezvous Regions: A scalable architecture for service location and data-xentric storage in large-scale wireless networks," in ACM Mobicom, San Diego, Sept. 2003.

[17] M. Klein, B. König-Ries and P. Obreiter, "Service Rings - A semantic overlay for service discovery in ad hoc networks," in International Workshop on Network-Based Information Systems (NBIS), Sept. 2003.

[18] S. Cirani, L. Davoli, G. Ferrari, R. Léone, P. Medagliani, M. Picone and L. Veltri, "A scalable and self-configuring architecture for service discovery in the internet of things," IEEE Internet of Things Journal, vol. 1, no. 5, pp. 508-521, 2014.

[19] P. Gomes, E. Cavalcante, T. Rodrigues and T. Batista, "A federated discovery service for the Internet of Things," in Workshop on Middleware for Context-Aware Applications in the IoT, Dec. 2015.

[20] Q. He, J. Yan, Y. Yang, R. Kowalczyk and H. Jin, "A decentralized service discovery approach on peer-to-peer networks," IEEE TSC, vol. 6, no. 1, pp. 64-75, 2013.

[21] M. Castro, P. Druschel, A. Kermarrec and A. Rowstron, "One ring to rule them all: Service discovery and binding in structured peer-to-peer overlay networks," in SIGOPS European Workshop, 2002.

[22] C. Schmidt and M. Parashar, "A peer-to-peer approach to web service discovery," vol. 7, no. 2, pp. 211-229, 2004.

[23] E. Adar and B. Huberman, "Free riding on Gnutella," First Monday, vol. 5, no. 10, 2001.

[24] J. Li and S. Vuong, "A scalable semantic routing architecture for grid resource discovery," in ICPADS, July, 2005.

[25] G. Koloniari and E. Pitoura, "Content-based routing of path queries in peer-to-peer systems," in International Conference on Extending Database Technology, 2004.

[26] J. Zhang, R. Shi, W. Wang, S. Lu, Y. Bai, Q. Bao, T. J. Lee, K. Nagaraja and N. Radia, "A Bloom filter-powered technique supporting scalable semantic service discovery in service networks," in ICWS, 2016.