A Web of Things based device-adaptive service composition framework

Michael Shell
School of Electrical and
Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332–0250
Email: http://www.michaelshell.org/contact.html

Homer Simpson
Twentieth Century Fox
Springfield, USA
Email: homer@thesimpsons.com

James Kirk
and Montgomery Scott
Starfleet Academy
San Francisco, California 96678–2391
Telephone: (800) 555–1212
Fax: (888) 555–1212

***Abstract*—In WoT environment, smart things provide RESTful services to expose their resources and operations. There are lots of smart things that offer the same functionalities but have different service interfaces. Because of the high coupling between device service instances and process specifications like BPEL, the cost of reusing a BPEL specification between different device environments is very high. We propose a device-adaptive service composition framework for WoT environment, in order to help users to apply the business process and service composition technologies more conveniently. In the framework, we design an activity description model, which is a semantic description for business activities, to overcome the shortcoming of directly binding the process and the service. Then, a matching mechanism between the model and the WADL of device services is proposed to select candidate services for the composition. Furthermore, we represent the matching result in a general service model, with which the source code of the general service can be automatically generated. The general service is a unified encapsulation for device services that match the functionalities of business activity. So user can interact with the general service instead of the origin services on the device, which decouple the process specification and the actual device services. A case study is offered to illustrate how to apply our framework in a intelligent charging pile sharing platform.**

***Keywords*—WoT;service composition;**

I. Introduction

During the last decade, the Internet of Things(IoT) technologies have been much more widely applied in both enterprise manufacturing and people's daily life. Concepts like Smart City, Intelligent Manufacturing, E-health are all closely bound up to IOT technologies. As increasing numbers of IoT devices (e.g., sensors, actuators, embeded devices, RFID) are getting connected to the World Wide Web(WWW)[1], the Web of Things(WoT) architecture offers us a new platform which enables us to manage and integrate the smart things in a much easier way with looser coupling and better scalability.

Different from the IoT that focuses on the network layer connectivity between physical devices in various constrained networking environments, the WoT focuses on integrating smart things into the application layer, the top level of the Internet. Web servers are embeded into smart things and the physical resources are exposed to the Internet as RESTful Web Services[2]. Compared with the traditional heavyweight Web Service technologies based on standards such as SOAP, WSDL, and UDDI , RESTful Web Service can be accessed in a easier way through HTTP or HTTPS protocol. In the RESTful architecture, all the resources are identified and encapsulated uniquely through the Universal Resource Identifiers(URIs), which can effectively solve the problems caused by the variability of data structure and data transmission mode between different devices and make the information exchange become more efficient through different programming languages, different platforms, and different devices[3].

As massive smart things are accessable in WoT environment, simple service invocation can hardly meet the requirement of increasingly complex applications. Therefore, our demand for business process management and service composition technologies in WoT environment is becoming very urgent. Business Process Execution Language(BPEL) is one of the most frequently used process definition standard[4]. BPEL binds business activities and Web Services and control the business processes with workflow, so as to execute complex business processes by service composition. However, such bindings refer actual service instances instead of service interfaces during the process definition phase, which will make high coupling between processes and devices in WoT. There are lots of smart things that offer the same functionalities but were produced by different manufacturers, which means that the Application Programming Interface(API), parameters, and constraints of those smart things are different. For example, there are hundreds of different models of charging pile produced by nearly a hundred charging pile manufacturers. Because of the variety of the device interface, it's very complicated to reuse a BPEL specification between different device environments, and a simple replacement of devices may lead to the rewriting of the entire specification.

In order to solve the defects above and allow users to make use of the business process and service composition technologies more conveniently, we propose a device-adaptive service composition framework for WoT environment. In the framework, we design an activity description model and a general service model. In the activity description model, by using ontology to construct a semantic extension of a business activity in BPEL, we can overcome the shortcoming of directly

binding the process and the service and make the process specification more reusable with the more abstract definition. Then, a matching mechanism between the model and the Web Application Description Language(WADL) is proposed to select available services from all the services of the device. The matching result is represented by a general service model, which contains the references to actual service instances. According to the general service model, source code of the general service can be automatically generated. The source code will be deployed on the device to publish the composited service for the business activity, so that users only need to interact with such service during the process execution and should pay no attention to the origin services on the device, which decouple the process specification and the actual device services. Through this approach, once the business activity should be reused in other device environments, we can generate the general services on the new devices automatically instead of redefine the entire process.

The remainder of the paper is organized as follows. In Section II, we introduce some existing semantic related business process and service composition technologies, and make a comparision with our approach. The overall architecture of our framework is briefly presented in Section III. In Section IV, we describe our activity description model, general service model, algorithms for matching and code generation in detail. Then, there is a case study about applying our framework in a scene of intelligent charging piles in Section V and we draw our conclusions in Section VI.

II. Related Work

To integrate traditional enterprise services and wireless sensor networks, [6] proposes a lightweight approach following the success of Web 2.0 ”mashups, proposing and implementing a collection of RESTful services to expose sensor nodes to web resources so that all the nodes become part of a Web of Thing and interact with existing nodes to compose their services. [7] combines the above work with an AutoWot platform, which offers a generic way to model Web resources and build Web components in order to facilitate the integration of smart devices into web.The integration, however, is based on particular service instances for service composition, failing to describe the composition from a abstract level.

[8] designs a WebPlug framework to strengthen the combination of real physical devices with virtual resources. In the framework, users can get access to physical resources by URL as well as the MetaURL defined by framework to obtain the related information about the device. For example adding @history after the URL of the device can archive the history information of this resource.SemSense proposed by [9] is composed of data collection module, storage module, semantic module and release module. Data are first collected from physical sensor, enriched by semantic annotation using LinkedData and finally published. Both approaches above are limited to annotating the resources, and are lack of the coordination among devices and services.

For the semantic extension of process description language, traditional methods like BPEL have a low precision about the similarity between conceptual similar service, due to the strict matching strategy[14]. WSDL-S[15] can’t distinguish between service type(concept) and service instance(individual), as it’s restriction in the expansion of the service concept. Likewise, in OWL-S [16], there is no significant difference between the process instance (the actual combination of services) and conceptual process (only relating to the concept of services).

In [19], researchers propose a RESTful service composition modeling method based on UML, using a conceptual resource model to describe the static composition structure. Besides, a state machine is used to represent the behavior information of the HTTP methods like PUT and POST. But the drawback is that it is hard to implement the actual service composition based on the model.

[20] proposes an approach termed as Web Service Resource Bundle(WSRB), considering the dependencies between the candidate services and binding those services during the invocation from the client, which offers a novel runtime compositing concept. But WSRB focuses on the fixed and predefined composition, failing to support the dynamic composition based on the business requirement.

The above literature survey shows that it is hard to directly apply existing business process and service composition technologies to WoT.

III. Framework

The overall architecture of our framework is shown in Fig. 1. In a business process, each activity is bound to an activity description model. The model is defined in a XML structure by RDF, which is a recommended semantic modeling

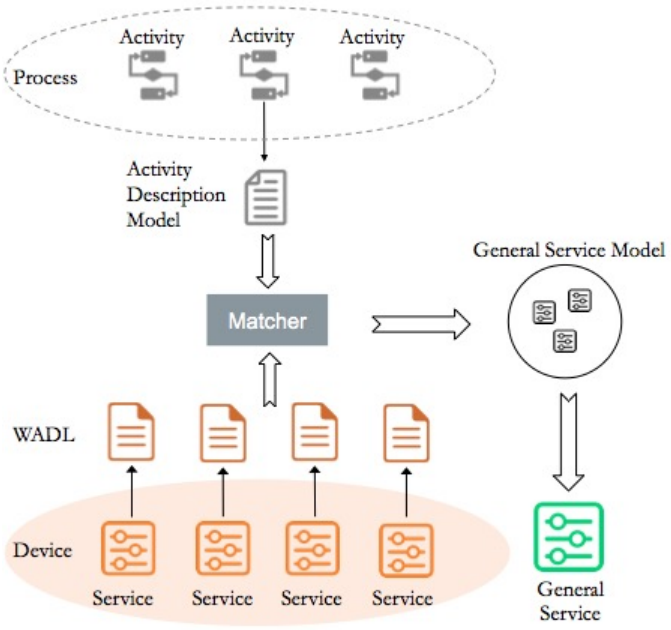

Fig. 1. Overall architecture for the framework.

metadata. The model provides ontological annotations for the binding activity abouts it's functionalities and messages. In order to reduce extra work for modeling, an Eclipse plugin is offered to help users to construct the models more effectively after the process is defined.

The matcher is the central component. Once the device envrionment is changed, users can upload the WADL description of the services of the new device and the activity description model to the matcher. A semantic-based matching mechanism is executed to find suitable service composition strategy, and the matching result will be presented in a general service model.

The general service model is consist of two part: (1) description of the general service, including resource path, request method, request and response parameters, and (2)actual service instances to composite and parameters mappings. On the basis of the model, a general service can be automatically generated. For the invoker of the general service, the service provides a device-independent request interface, which means that the invoker is able to call the general services from different devices that have the same capabilities of the process activity and don't need to modify the HTTP request at all.

IV. Approach

A. Activity Description Model

When expressing a business process in BPEL, we are essentially defining a composite relationship of existing services. In BPEL, the $< invoke >$ primitive is used to represent the common task that invoke a Web Service instance. Nevertheless, in WoT, such kind of direct service binding may lead to several limitations. Because device environment is changeable and devices with similar functionalities offer different services, the cost of maintaining the process specification is very high.

In the context of intelligent charging pill, the capabilities of the devices are exactly the same, but the process definition cannot be shared. The invoke bindings of the process must be rewritten to adapt the service interface of different charging pill.

To address such limitations, a more abstract and more generate workflow definition is a feasible solution. We propose an activity description model, to attach ontological description for the business activity. The model can be represented as a tuple,

$$AD = <\mathbf{DM}, \mathbf{IM}, \mathbf{OM}, \mathbf{BF}> \quad (1)$$

where **DM**, **IM**, **OM**, **BF** denote domain, input messages, output messages, and business functionalities. Domain represents the device environment of current activity. Input messages and output messages contain variables with identity and data type. Business functionalities are consist of several verb-object constructions, which describe the business activity from a resource and operation perspective.

In lots of semantic based service composition technologies, researchers perfer to take advantage of Input, Output, Pre-condition, Effect(IOPE) as the basis of service description and the critical parameters of the service matchmaking calculation[5][6]. However, in our model, we choose the simpler verb-object construction instead of the Pre-condition and Effect(PE), because: (1) Most of the RESTful Web Services in WoT are simple resource operations; (2) Defining a PE is much more sophisticated than defining a verb-object construction; (3) To campare with the semantic similarity calculation(between the functionality and the resource URL), the matchmaking calculation on PE is inefficiency.

Taking the charging pill as an example, an activity description model instance about the "reserveCharging" activity is shown in Fig. 2. The core resource operation of this activity is creating a reservation for a charging pill, which takes $startTime$ and $endTime$ as constructor arguments, and $result$ and $detail$ are returned to show whether the create operation is success or not and a detailed reason for the failure.

Also, we try to provide a convenient and efficient way for users to specify their workflow in our model. As is shown in Fig. 3, a graphical modeling tool is afforded in the form of an Eclipse plugin. With such plugin, users can create an activity description model file for a business activity, drag and drop model nodes to the canvas, and fulfil the properties in an editing frame, just like defining a BPEL schema with the BPEL Designer for Eclipse.

B. Matching Strategy

In this subsection, we will give a detailed introduction to our matching strategy between our activity description model and device services. The WADL documents of those servics will be utilized for seeking out appropriate services that match the business functionality description.

In actual Internet applications, capabilities provided by RESTful services are far more complex than pure resource operations, and the form of data exchange is more in nested structure like JSON and XML than basic data types like string or integer. But, as mentioned above, in the WOT environment, most RESTful services provided by smart devices are simple resource operations, which will not be too complicated. Therefore, our matching strategy will pay more attention to the

```
<Activity Class rdf:ID="reserveCharging">
    <Domain rdf:resource="#ChargingPill"/>
    <HasInputMessage>
        <InputMessage rdf:ID="startTime" rdf:datatype="Date"/>
        <InputMessage rdf:ID="endTime" rdf:datatype="Date"/>
    </HasInputMessage>
    <HasOutputMessage>
        <OutputMessage rdf:ID="result" rdf:datatype="Boolean"/>
        <OutputMessage rdf:ID="detail" rdf:datatype="String"/>
    </HasOutputMesage>
    <HasFunctionality>
        <BusinessFunctionality rdf:ID="createReservation">
            <Operation rdf:ID="create"/>
            <Resource rdf:ID="reservation"/>
        </BusinessFunctionality>
    </Business>
</Activity Class>
```

Fig. 2. Sample description of a reservation activity.

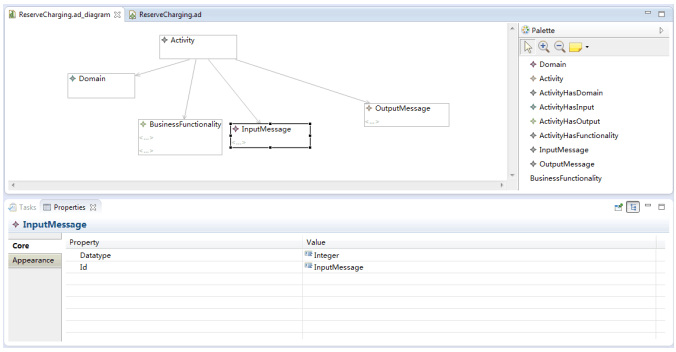

Fig. 3. A graphical modeling tool for the activity description model.

```
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://wadl.dev.java.net/2009/02 wadl.xsd"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns="http://wadl.dev.java.net/2009/02">
  <resources base="http://api.search.yahoo.com/NewsSearchService/V1/">
   <resource path="newsSearch">
    <method name="GET" id="search">
     <request>
      <param name="appid" type="xsd:string" style="query" required="true"/>
      <param name="query" type="xsd:string" style="query" required="true"/>
      <param name="type" style="query" default="all">
       <option value="all"/>
       <option value="any"/>
       <option value="phrase"/>
      </param>
     </request>
     <response status="200">
      <representation mediaType="application/xml" element="yn:ResultSet"/>
     </response>
     <response status="400">
      <representation mediaType="application/xml"  element="ya:Error"/>
     </response>
    </method>
   </resource>
  </resources>
</application>
```

Fig. 4. Sample description of a reservation activity.

equivalence of resources and operations, so that the complexity of the matching can be reduced to get a higher matching efficiency.

Fig. 4 shows a common WADL file example from Wikipedia. Generally speaking, a RESTful service denote an operation to a resource. Information about what resource the service has and how to invode the service is contained in the WADL document throught following nodes: (1) A *resources* node is seen as a container for the *resource* nodes, presenting the basic URL address of it's resources; (2) A *resource* node specifies the resource to operate in the service with a path attribute; (3) A *method* node is embodied in a *resource* node to show the HTTP request method (e.g., GET, PUT, POST, DELETE) that can be accepted by the service, which also stand for the operation towards resource; (4) A *request* node is a container for the *param* nodes; (5) A *param* node describes an input parameter of the service, including the data type of the parameter and whether such parameter is necessary or not.

According to the WADL structure above, we can find that information about resource and operation of a service is in the *resource* node and the *method* node. In the *resource* node, the path attribute is generally constitutive of a group of simple words, such as user/register and user/password/reset. Such group of words can always be divided into two part, a series of nouns to locate the URI of the resource, and a verb to appoint the operation on it.

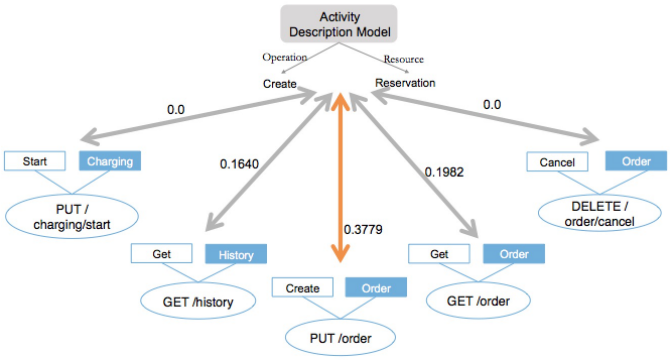

Fig. 5. Matching result of a charging pile example.

However, in some special cases, there might be no verb in a URL path. We often use a user/{userid}/orders path to indicate the action that fetch all the orders of a user. In this circumstance, the operation information should be complemented by the HTTP request method in the *method* node. There are eight HTTP request methods defined in the HTTP/1.1 specification, while the GET, PUT, POST, DELETE methods are the most commonly used ones. Owing to the HTTP specification, such methods have well known and dependable semantic meaning. A request with the GET method is always used for retrieving data. Consequently, for URL like user/{userId}/orders with the GET method, we can infer that it is a query service and the operation on the resource can be annotated as a "search" or a "get". Similarly, with a PUT or POST method, we can associate a "create" operation with a resource that have no specified identifier (e.g., userId), and an "update" operation with a resource that have specified identifier.

After recognizing the resource and the operation of a service, we can refine the semantic similarity between the service and the business functionality defined in our activity description model. The similarity calculation formula is:

$$Sim(f, s) = S(r_f, r_s) \times S(o_f, o_s) \quad (2)$$

In (2), f and s denote the functionality(described in the model) and the service. $S(r_f, r_s)$ is the similarity between the resources in the functionality and the service, which can be predefined in an ontology repository or be retrieved in the WordNet[7].

Due to the limitation of performance in smart devices, we can assume that there will not be two services in one device that offer resemble operations on the same resource. Therefore, a conclusion can be drawn that the service with the highest similarity is the matching service. The algorithm for finding the matching service is defined in Algorithm 1.

A prototype application of the algorithm is implemented. Since there is no open collection of WoT services to verify the

Algorithm 1 Find Matching Service

Input: resource and operation of a functionality description R_f, O_f, and services of a smart device $S = \{s_1, s_2, \cdots, s_n\}$

Output: the matching service to the functionality description

for each service s_i in S **do**
  Let W_i be the WADL description of s_i
  Let $path$ be the path attribute of the resource node in W_i
  Split $path$ to get a word vector $WORD_i$
  Let R_{si} and O_{si} be the last noun and verb in the vector $WORD_i$
  if O_{si} is null **then**
    Complement the O_{si} based on the $method$ node in W_i
  end if
  $SIM_{si} \leftarrow Sim(R_f, R_{si}) \times Sim(O_f, O_{si})$ //$Sim(a, b)$ is the API to calculate the similarity between a and b
end for
Let k be the index of the maximum similarity in SIM
return service s_k

precision of our matching, we have made a simple example for testing. Some RESTful services of the intelligent charging pile is obtained from an existing platform, and a matching result is shown in Fig. 5. By using the WordNet, we calculate the similarity between five services and our functionality defined in Fig. 2. The "PUT /order" service is choosed to be the matching service because of the highest similarity, which is actually the operation to add a new reservation. However, there are several domain specified concepts that can not be retrieved in the WordNet, which may cause some wrong computing results. Therefore, a domain specified ontology repository is more suggested to optimize the matching accuracy.

C. General Service Model

After the matching service is found, the following question arises:"How to generate a device-independent service with services on the device".

An important factor to solve this question is to find the corresponding relation between the message description of the activity and the parameters of the services. For services with similar capabilities in various equipement environments, the form of passing arguments might be different. As an example, when querying for the same resource, the name and order of the query conditions might be different in different devices.

Nevertheless, even if the representation is different, the data type and the semantic meaning of those parameters are consistent. Since the structures of the parameter description in our model and WADL are both consist of a name attribute and a data type attribute, the semantic based matching method is still a feasible solution. The semantic similarity calculation

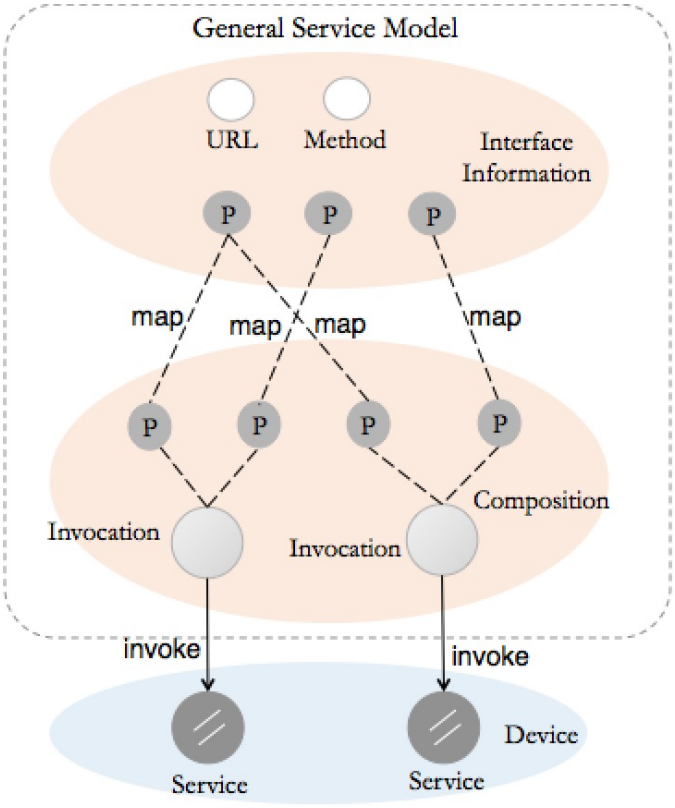

Fig. 6. Conception of the general service model.

formula between two parameter descriptions is:

$$Sim(P_1, P_2) = \begin{cases} 0 & \text{if } t_1 \text{ and } t_2 \text{ are not analogous} \\ S(n_1, n_2) & \text{if } t_1 \text{ and } t_2 \text{ are analogous} \end{cases} \quad (3)$$

where t_1 and t_2 are the data types of parameter description P_1 and P_2, and n_1 and n_2 are name of two parameters. A metrix can be predefined to check whether two data types are analogous(e.g., string and varchar are analogous but integer and date are not).

As the matching degree of two parameter description is measured, we can now obtain the parameter matching result between our activity description and practical device services.

In order to generate the device-independent service, we propose a general service model to express the composition of the services that match the business functionalities in an activity.

A basic concept of the general service model is shown in fig. 6. Structurally, the model is organized in two parts. The first part is the interface information of the general service, which defines the API for the composited service that fulfil the business activity. Considering the device-independency, the interface information is generated from the activity description model, including the URL path, the HTTP request method and the description of request and response parameters. The other one is the composition part. In this part, candidate services are listed with details about how to invoke the service and where

to find the parameters.

Algorithm 2 Generate Service Code

Input: a general service model, including URL, $method$, input parameters $IP = \{ip_1, ip_2, \cdots, ip_n\}$, output parameters $OP = \{op_1, op_2, \cdots, op_n\}$ and invocations $Inv = \{inv_1, inv_2, \cdots, inv_n\}$

Output: source code of the service

1: generate the service interface with URL and $method$
2: **for** each parameter ip_i in IP **do**
3: set up the mapping between input ip_i and content in URL
4: **end for**
5: **for** each invocation inv_i in Inv **do**
6: generate a new HTTP request with interface in inv_i and parameters in IP
7: set up the mapping between output OP and content in HTTP response
8: **end for**
9: **return** the source code of the service

Assume we need to execute an activity on two different devices. For each device, a general service model is generated. Comparing two models, we can find that the contents of their interface information part are exactly the same, to make the same HTTP request can be directly reused in both devices. On the contrary, the composition parts on two models are device specific, to composite the actual services on the device.

Since the general service model furnish an abstract description about the service composition, the specific JavaScript code can be generated to implement and deploy the general service on a Node.js server[8]. A rough process for code generation is introduced in Algorithm 2.

The general service provides a device-independent interface to decouple business code from specific device service. Business activity can bind with the general service instead of the original services on the device. When there is an replacement between two different devices, the process can switch to the new device without any modification about the source code.

V. Case Study

In this section, a case study is introduced to verify the feasibility and effectiveness of the approaches above. We construct a prototype system, EVER, an intelligent charging pile sharing platform, to illustrate a typical application of our framework.

As a kind of WoT equipment, intelligent charging pile is connected to the Internet through WIFI or GPRS, offering RESTful services as operation interfaces. In China, charging pile sharing is an emerging area. The owner can make his private charging pile open to the public, and the public can charge their car by paying a certain fee to the owner. The sharing platform brings a win-win pattern, as the lack of public charging pile can be alleviated and the owner can cover the cost of building the charging pile through the income.

```
<GeneralService id="reserveCharging">
 <interface path="/reservation/create" method="PUT">
  <request>
   <param id="input1" name="startTime" type="date" required="true"/>
   <param id="input2" name="endTime" type="date" required="true"/>
  </request>
  <response>
   <param id="output1" name="result" type="boolean" required="true"/>
   <param id="output2" name="detail" type="string" required="false"/>
  </response>
 </interface>
 <composition>
  <invocation url="/order" method="PUT">
   <request>
    <param name="begin" type="date" required="true" mapping="input1"/>
    <param name="end" type="date" required="true" mapping="input2"/>
   </request>
   <response>
    <param name="result" type="boolean" required="true" mapping="output1"/>
   </response>
  </invocation>
 </composition>
</GeneralService>
```

Fig. 7. An example of general service model.

A brief process of charging through the sharing platform has following steps:

- When a driver need to charge his car, he can open the App on his smart phone to fetch all the candidate charging piles nearby.
- Then, he can choose a pile and check the detail information of it, including suitable car type, available time, charging fee and etc.
- If the pile is suitable, he can make a reservation in his App.
- After arriving the charging pile, he can scan the QR code on the charging pile to start charging.
- When the charging is over, he can stop the charging in his App and get the bill to pay.

During the charging process, several device services are used to perform business functionalities such as making reservation and generating bill. However, as there are hundreds of different charging pile models in the market, how to adapt the business process to all those models is a great challenge.

Based on our framework, we designed and implemented a sharing platform, EVER, which has a strong adaptability for the cross-devices process reuse.

A. Application of the framework

Reviewing the business process of charging, we found that most activities in the process are related to one or more device services. Therefore, with the help of our Eclipse modeling plugin, we have described those activities in the activity description model. An example of those descriptions is introduced in Fig. 2.

When a pile owner want to register and share his pile on our platform, the WADL documents of device services should be uploaded. With those descriptions, our matching algorithm is exectued to generate the general service models. Fig. 7 serves as an example of a general service model. In the model, the $invocation$ node presents a device service interface for the

functionality of reserving charging, which is bound to the general ”/reservation/create” service.

Based on the sample model, source code of the general service can be generated, which is shown in Fig. 8. After the general services are deployed on the pile, the pile is now available to the public.

When a user calls a general service, the general service will help the user to invoke real services on the device. Consequently, our general service offers a device-independent interface to users, which hide all the device services from the user.

B. Implementation of the system

EVER is consist of four parts: center server, web site, App and intelligent charging pile. The pile provides RESTful services for the device resources. The center server is in charge of the other APIs of the system. The web site offers user interface for the management of the intelligent charging piles, such as updating the pile state or checking the usage record. Fig. 9 presents the web page for sharing a new charging pile. The App provides the main charging business. Users can make the reservation, control the charging and pay for bill on the App. The reservation and charging pages of the App is shown in Fig. 10.

```
app.put('reservation/create',function(req, res){
  // the api options
  var options = {
    host : '192.168.1.1',
    port : '8080',
    path : '/order?begin='+req.query.startTime+'&end='+req.query.endTime,
    method : 'PUT'
  };
  var reqPut = http.request(options, function(resPut) {
    resPut.on('data', function(d) {
      if(d.statusCode == 200) {
        var data = {result:d.result, detail:''}
        res.send(200, JSON.stringify(data));
      }
      else {
        res.send(500, "error");
      }
    });
  });
  reqPut.end();
});
```

Fig. 8. Source code of a general service.

Fig. 9. A screenshot of the prototype system.

Fig. 10. Screenshots of the App.

VI. Conclusion

In this paper, we propose a device-adaptive service composition framework for WoT environment, to avoid the high cost of reusing a BPEL specification between different devices. In the framework, an activity description model is proposed to attach semantic meaning to business activity. After matching and generating, a general service is deployed to provide a unified interface, which achieve the business activity by compositing the device services. So that users only need to interact with the general services during the process execution and should pay no attention to the origin services on the device, which decouple the process specification and the actual device services.

In the future, we will try to construct a WoT ontology repository to improve our matching performance. Besides, we will extend our framework so as to record the status of executing processes, so that the devices can be replaced dynamically.

Acknowledgment

The authors would like to thank...

References

[1] D. Guinard, V. Trifa, F. Mattern, and E. Wilde, “From the internet of things to the web of things: Resource-oriented architecture and best practices,” in *Architecting the Internet of Things*. Springer, 2011, pp. 97–129.

[2] D. Guinard, V. Trifa, and E. Wilde, “A resource oriented architecture for the web of things,” in *Internet of Things (IOT), 2010*. IEEE, 2010, pp. 1–8.

[3] C. Pautasso, O. Zimmermann, and F. Leymann, “Restful web services vs. big’web services: making the right architectural decision,” in *Proceedings of the 17th international conference on World Wide Web*. ACM, 2008, pp. 805–814.

[4] D. Jordan, J. Evdemon, A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland *et al.*, "Web services business process execution language version 2.0," 2003.
[5] Y. Syu, S.-P. Ma, J.-Y. Kuo, and Y.-Y. FanJiang, "A survey on automated service composition methods and related techniques," in *Services Computing (SCC), 2012 IEEE Ninth International Conference on*. IEEE, 2012, pp. 290–297.
[6] C.-H. Lee, S.-Y. Hwang, and I.-L. Yen, "Service composition with functional flexibility using nondeterministic service interface," in *e-Business Engineering (ICEBE), 2013 IEEE 10th International Conference on*. IEEE, 2013, pp. 435–440.
[7] "Wordnet," http://wordnet.princeton.edu/.
[8] D. Raggett, "Web of things framework," https://www.w3.org/2015/05/wot-framework.pdf.