IoT-MAP: IoT Mashup Application Platform for the Flexible IoT Ecosystem

Sehyeon Heo, Sungpil Woo, Janggwan Im, Daeyoung Kim
Department of Computer Science,
Korea Advanced Institute of Science and Technology
Daejeon, South Korea
crux042, woosungpil, limg00n, kimd@kaist.ac.kr

***Abstract*—Recent mobile environment has various types of smart things which are opportunistically adjacent to the surrounding areas of mobile phones. However most of applications in the market are just statically bound to specific model of designated manufacturer which were chosen in development phase, so they are not interoperable with heterogeneous things even though they have similar functionalities. To optimize utilization of smart things residing in each user's personal pervasive environment, IoT mashup application is required which dynamically discovers heterogeneous things, downloads needed software module at runtime, and provides mash-up application to the user. In this paper, we designed and implemented IoT Mashup Application Platform that supports smartphone-centric discovery, identification, installation, mashup and composition of the pervasive smart things. Furthermore, by decoupling the roles of actors in IoT ecosystem utilizing functionally abstracted thing interfaces, the platform provides benefits to each actor - thing manufacturers can maximize compatibility of their products, application developers can concentrate on their own business logic, and the end-users can select which smart things would be actually used for the composition.**

***Keywords*-Internet of Things; Service Composition; Mashup; Interoperability; EPCglobal**

I. Introduction

Numerous types of smart devices are coming into the IoT ecosystem these days, and some of them are even becoming integral parts of our everyday life. In this situation, if the flexible interaction between such smart things and personal mobile device become feasible, they can provide better user experiences and novel services compared to the stock applications that only interacts with specific devices. For example, we can think of a simple service that visualizes body temperature or heart rate measured with body-attached sensors, by controlling brightness or color of a smart bulb. In conventional mobile development environment, applications that provide such services are usually bound to particular model of things which was chosen by developer at implementation time. Such applications could interact with only the smart things whose binding has been defined at implementation time. So, current mobile applications cannot easily adapt to the surrounding environment.

In this paper, we propose IoT-MAP, a mobile application platform aiding flexible interoperability between mobile devices and surrounding smart things - and we name those

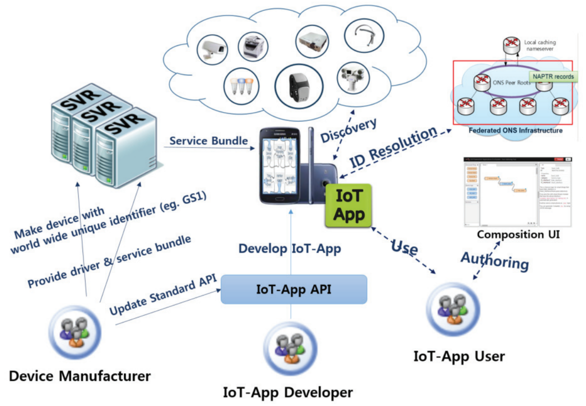


Figure 1: High-level Concept Diagram

applications developed based on the platform library as IoT-App. The IoT-MAP platform provides various improvements of actors in current IoT ecosystem as shown in Figure 1. For the mobile application developers, IoT-MAP provides a set of intuitive APIs to build an IoT-App easily, such as thing discovery, connection and retrieval of abstracted service object which can be directly used as if it is actual thing. With those support of IoT-MAP platform, they can write their own business logic in POJO (plain old java object) style without consideration of actual connectivity and implementation of smart things. For the thing manufacturers, they can easily participate in the ID resolution procedure of the platform by preparing their own name server based on ONS and driver bundle repository. Because the ONS is designed based on DNS technology, the manufacturer's name server can be integrated to other distributed ONS servers, therefore the end-users can find driver bundle of their things directly provided by the manufacturer without any knowledge (e.g. manufacturer name, thing name, serial number, etc.) of the thing. And finally for the end-users, they can choose which thing would be actually participate in the desired service by selecting the discovered things at run-time. Moreover, in case of there is no IoT-App matches user's requirement in the market, end-users can utilize the GUI authoring tool provided by IoT-MAP platform. The users can assemble application using their finger at run-time with discovered things and logic modules that make each things interoperable, then those modules are dynamically downloaded and composed to form an application of user's purpose.

The contribution of this paper is summarized as follows. First, we propose the IoT-MAP, a novel IoT mashup application platform that provides flexible interaction with surrounding things. Second, we designed an international standard based Thing ID resolution procedure utilizing well-established name server implementation - Oliot-ONS (Object Name Service) which is a component of IoT infrastructure platform Oliot(http://www.oliot.org). Third, we provide two ways of development methods - IoT-App Library for application developers to create IoT-App, and GUI author tool for end-users to assemble application fit on their purpose at run time intuitively. Finally, we report experimental results on overhead evaluation of IoT-MAP and give two demo scenarios utilizing the platform for verification.

The rest of this paper is organized as follows. In Section 2, we describe about high level concept and design consideration of our IoT-MAP platform. Section 3 introduces architecture of IoT-MAP to support development and operation of IoT-Apps, and Section 4 explain critical requirements for dynamic deployment and service composition of smart things and how they are handled by our platform. Then we present implementation in Section 5, show evaluation and demo application scenarios for verification In Section 6. We introduce related work in Section 7 and finally conclude the paper in section 8.

II. Design Considerations

Many IoT devices are coming to the market nowadays, but their capabilities are limited because they are equipped with a mobile application which was specifically designed for the devices. For example, smart bulbs such as Philips Hue (http://meethue.com) provide a mobile application to turn on/off or change colors. End-users cannot customize use of smart bulbs according to other application logics, sensors or actuators interactions.

The problem is that the role of device manufacturers is not differentiated with that of application developers. Application developers focus on how to use the functionalities of heterogeneous devices with IoT-App API, regardless of underlying connectivity protocols while device manufacturers focus on providing the device functionalities correctly by implementing underlying connectivity protocols. Furthermore, due to variety of end-user preferences, end-users may want to create or compose their own or versatile applications at run-time using a graphical user interface (GUI). It means that the role of application developers may be extended to that of end-users. Therefore, a platform is necessary where application developers and device manufacturers concentrate on their role to satisfy variety of user requirements.

The concept of model driven architecture (MDA) [17] can provide a layered architecture so that division of roles among end-users, application developers, and device manufacturers is achieved. The idea of MDA is to extract platform-independent and domain-specific model (Platform Independent Model, PIM) from platform-specific elements such as programming languages, operating systems, or communication protocols (Platform Specific Model, PSM) to achieve an efficient development or interoperability between different systems.

Code 1: Example of Abstracted Interfaces and Functional Description of SensorTag

```
public interface ThingService {
    String getThingId();
    String getThingName();
    void setEndpoint(Endpoint endpoint);
    void connect();
    void disconnect();
    boolean isConnected();
}
public interface ButtonService
        extends ThingService {
    int getButtonCount();
    boolean getButtonState(int index);
}
public interface AccelService
        extends ThingService {
    String getAccelRateMeasurement();
}
public interface SensorTagService
        extends ButtonService, AccelService,
                GyroscopeService, ... {
}
```

In PIM layer of the platform, application developers access devices through well-defined interface regardless of underlying connectivity protocols. The platform provides generalized functional abstraction interfaces which is not constrained by vendor-specific one, and can also be composed to describe functionalities of specific things like Code 1. The *ThingService* describes base functions of physical things such as retrieving identification, setting endpoint and connect/disconnect. *ButtonService* and *AccelService* describes atomic functions to poll simple button data and accelerator data. And with these interfaces *SensorTagService* can be written to describe functionalities of a specific model of thing, TI SensorTag. With these abstractions of device functionalities, potential of IoT-App can be drastically increased because device manufacturers, application developers, and end-users are all interoperable.

Due to the presence of PIM layer, it will be possible for end-users to create their own or versatile applications at run-time. In the layer above PIM layer, a GUI tool can be deployed which provides a list of discovered devices and available software components of application logics so that end-users assemble them to create applications dynamically. This is possible because PIM layer already abstracted device and underlying communication protocols.

In PSM layer in the platform, the functionalities of devices

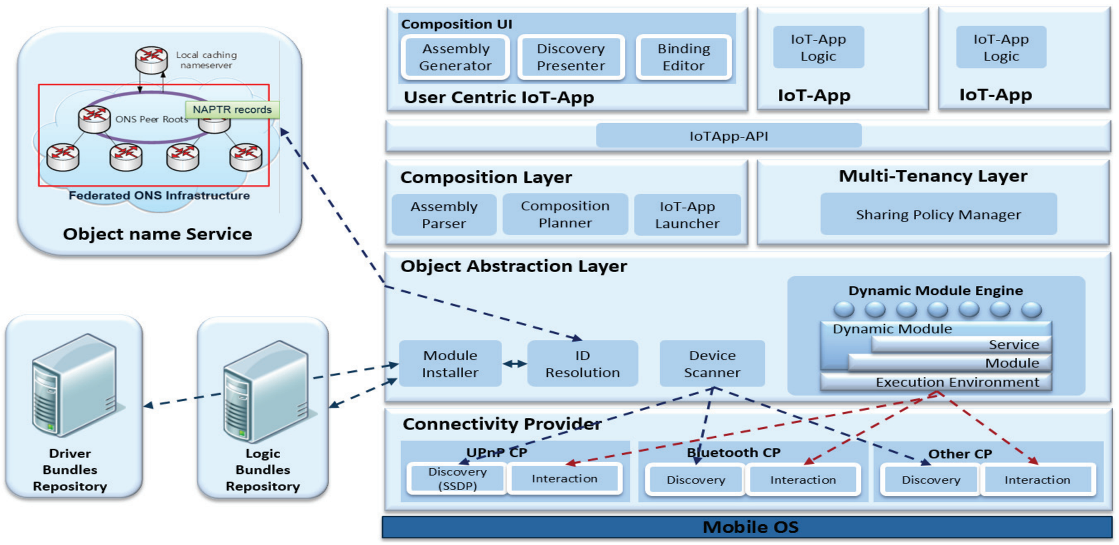


Figure 2: IoT-MAP Platform Architecture

can be provided by device manufacturers implementing well-defined interface and connectivity protocol. Note that the same functionality of the device can be implemented with different communication protocols. For example, smart bulb interface such as turning on/off, and changing color can be implemented either by Bluetooth or UPnP over WiFi. This choice is totally dependent on device manufacturer's decision, but applications can still access the device functionalities using the same interface.

For the run-time loading of platform-specific implementations, external repository is necessary. It should not only provide retrieving function of implementation codes according to the identifiers of the devices but also be distributed so that scalable searching is possible. We exploited object name service (ONS) infrastructure from GS1 EPCglobal architecture. It uses the identification system called electronic product code (EPC) and leverages the hierarchical structure of domain name service (DNS), so scalable global-scale distribution and efficient retrieving of platform-specific implementation codes is possible.

III. The IoT Mashup Application Platform

We design IoT-MAP architecture that works on mobile operating system and could be used for developer and user. Developers could build the application that interact with various smart things easily by using IoT-App API that utilize thing's functionalities in uniform way if you have IoT-MAP abstract functionalities of various heterogeneous smart things around user's smartphone. For users, they could simply use IoT-App implemented by developer by selecting specific things discovered by the platform at runtime or they could author IoT-App at run time by using Composition UI (GUI tool for user). IoT-App can interact with various things without static binding to specific vendor at implementation time, where as Normal application can only interact with things that are bounded at implementation time.

A. *IoT-MAP Platform System Architecture*

The platform abstracts basic functionalities of device OS (e.g. connectivity, network, etc.) so that IoT-Apps can utilize them. Figure 2 is the full stack diagram of the platform. The bottom-most part of the platform, Connectivity Provider provides various connectivities and protocols to be used for discovering and communicating with smart things around. And the Object Abstraction Layer is in charge of discovery, identification, virtualized object management and bundle management, and actual composition of smart thing services and business logic. Composition Layer supports functionalities for Versatile App. This Layer actually carries out the service composition by authoring the parsing information that is defined by user at run-time on authoring tool and retrieving the reference of bundles from the object abstraction layer. And finally, there is an IoT-APP API which offers standard interfaces for smart-things' group (e.g. bulbs, sensors, cameras, etc.) and also offers API for the various functions such as the things discovery, retrieve virtualized object and etc.

1) Connectivity Provider: Connectivity Provider abstracts and provide various connectivities and protocols to upper layer of platform, like Bluetooth or Bluetooth Low Energy (BLE) supported by Android, or other protocols such as UPnP supported by external libraries. Developer could discover external things by using API for the things discovery. Bluetooth CP (Connectivity Provider), UPnP CP, and other CPs are implementing *ConnectivityProvider* interface, so upper layer can utilize basic functionalities such as discovery and retrieval of identification or profile of things that support each CP in general way.

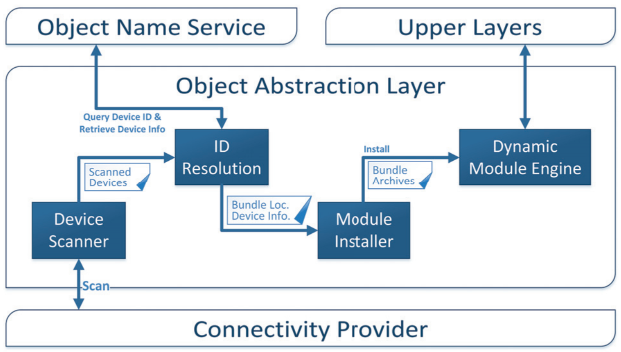


Figure 3: Structure of Object Abstraction Layer

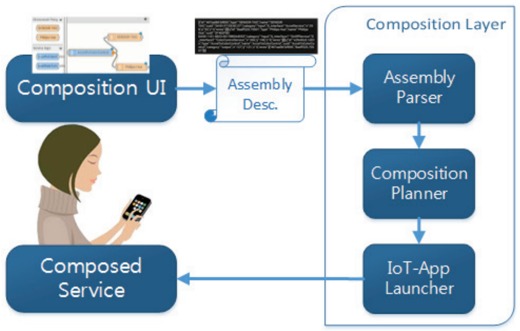


Figure 4: Structure of Composition Layer

2) *Object Abstraction Layer:* Object Abstraction Layer in Figure 3 is a core layer of IoT-MAP Platform, which abstracts real-world devices into group of abstracted services and enables composition of those services and business logic of IoT-App. This layer includes Device Scanner which discovers surrounding smart things, ID Resolution which identify and locate the device information, Module Installer which download and register device software module, and Dynamic Module Engine which dynamically manages (install, start, stop, update, uninstall) devices as abstracted service object with the aid of OSGi framework. The discovered and available devices pass through these components and stored to the Dynamic Module Engine to be used as exported device service by upper layer of platform.

- **Device Scanner**
  Device Scanner scans available devices around the smartphone using discovery methods (e.g. SSDP for UPnP) of Connectivity Provider. To support scanning capabilities on heterogeneous connectivity and protocols, each of them are implemented in *strategy pattern* which defines several algorithm, encapsulates each one, and makes them interchangeable with uniform interface *DeviceScanner*. Each scanned device is described with *ScannedDevice* object that includes connectivity, protocol, device id, name, and endpoint.
- **ID Resolution**
  Smart things usually have identifiable ID such as URN, MAC address, UUID, or GS1 ID. ID Resolution component reads these identifiers from *ScannedDevice* object and query it to external Object Name Service server to locate device information and driver bundle URL. The procedure of this is handled in section IV.
- **Module Installer**
  Module Installer reads the location of driver bundle from the result of ID Resolution, and dynamically download it from bundle repository. Then it installs the bundle to Dynamic Module Engine.
- **Dynamic Module Engine**
  Dynamic Module Engine is a core engine of IoT-MAP Platform that is responsible of dynamically load various device drivers and service logics, and those modules are composed as parts of IoT-App. So this engine is implemented utilizing OSGi Framework[11] to consume its powerful implementations such as execution environment, module management, lifecycle management, and service management. The engine can provide the instances of software modules as service, and IoT-App can load these instances as if they are in a single application. Most complex procedures of this operations can be delegated to OSGi Framework.

B. *Composition Layer*

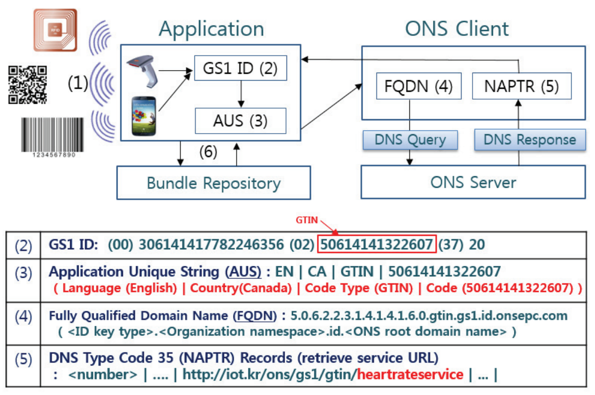
Composition layer is utilized by *Versatile App*, which is a special type of IoT-App that can discover and connect device by parsing the authoring information from User authoring tool. General IoT-Apps integrates smart things based on application developer's business logic, but Versatile App gathers devices from authoring information, and compose each software module based on that information. The structure of this layer is illustrated in Figure 4. Composition UI shows the list of service bundles as well as the available application logic bundle list in the user authoring tool. This plays the role of delivering the authoring information to the composition layer. Service bundle is a software module necessary to use the functions provided by the smart-things while the application logic bundle is a software module used to provide an application composed with the service bundle. When the user creates composition assembly, it is parsed, composed, and launched by this layer.

IV. Dynamic Identification and Deployment of Smart Things

To integrate heterogeneous devices in uniform way, IoT-MAP Platform provides genuine identification and deployment mechanism utilizing EPCglobal[1] standard.

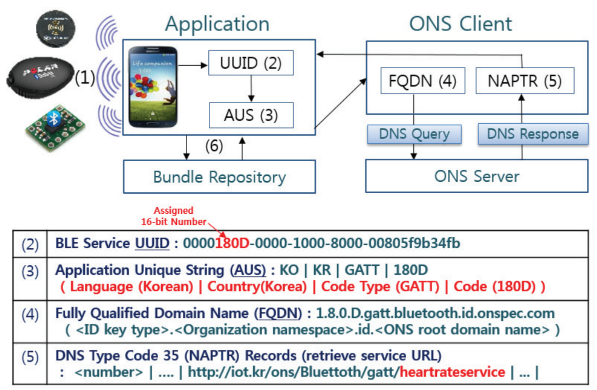
A. *EPCglobal and Object Name Service*

EPCglobal is set of standards for supply chain management, which is published and maintained by international standardization organization GS1. It defines ways to track



(2)	GS1 ID: (00) 306141417782246356 (02) 50614141322607 (37) 20
(3)	Application Unique String (AUS) : EN | CA | GTIN | 50614141322607 (Language (English) | Country(Canada) | Code Type (GTIN) | Code (50614141322607))
(4)	Fully Qualified Domain Name (FQDN) : 5.0.6.2.2.3.1.4.1.4.1.6.0.gtin.gs1.id.onsepc.com (<ID key type>.<Organization namespace>.id.<ONS root domain name>)
(5)	DNS Type Code 35 (NAPTR) Records (retrieve service URL) : <number> | .... | http://iot.kr/ons/gs1/gtin/heartrateservice | ... |

(a) Identification procedure of the products with assigned unique EPC



(2)	BLE Service UUID : 0000180D-0000-1000-8000-00805f9b34fb
(3)	Application Unique String (AUS) : KO | KR | GATT | 180D (Language (Korean) | Country(Korea) | Code Type (GATT) | Code (180D))
(4)	Fully Qualified Domain Name (FQDN) : 1.8.0.D.gatt.bluetooth.id.onspec.com (<ID key type>.<Organization namespace>.id.<ONS root domain name>)
(5)	DNS Type Code 35 (NAPTR) Records (retrieve service URL) : <number> | .... | http://iot.kr/ons/Bluettoth/gatt/heartrateservice | ... |

(b) Identification procedure of the device which supports BLE GATT profile

Figure 5: Device identification procedure with ONS

and identify each of the products throughout the whole factors of supply chain including production, distribution, and consumption. The core of this standard is the unique code EPC (Electronic Product Code) that is assigned to each of the products and that code connects actual product and virtual product data. Among the standards of EPCglobal, IoT-MAP Platform utilizes Object Name Service[2], which aid business stakeholders to retrieve information of that product from the server maintained by manufacturer. ONS is implemented based on Domain Name Service (DNS), so it can locate the information from wide-spread distributed servers maintained by each manufactures effectively.

Figure 5a illustrates the standard identification procedure of ONS with EPC. First, the user scan and discover thing with the scanner or smartphone(1, 2), read the EPC from it (3), then create Application Unique String (AUS) which contains user's environment information such as country and language code, and send it to ONS client(4). The ONS client converts the AUS to Fully Qualified Domain Name (FQDN) based on ONS standard, query it to local ONS server, then that ONS server locates the other server that contains thing information along the same lines as the DNS(5). The located ONS server generates Name Authority Pointer (NAPTR) record which is one of DNS response standards, then send it to the ONS client of user(6). And finally, the ONS client sends this information to user's application(7). Because the list of information is maintained by the manufacturer, it can be a nice method to deliver the location of device driver which resides in the manufacturer's server. With this environment, the IoT-MAP Platform can identify the smart thing, find location of driver, then download and install to deploy that thing into user's smartphone.

The powerful benefit of utilizing EPCglobal is that it is international standard for supply chain. As the GS1 barcode have been the standard for international trading, all products including various smart things can have unique EPC in the future. And if those things have EPC, then we can easily utilize that standard to identify and deploy the heterogeneous smart things in uniform way.

B. Electronic Product Code and the Alternatives

As mentioned in previous subsection, EPCglobal standard assigns unique code to each products. But obviously most of smart things does not have EPC as its identifier and those currently available in market usually have identifiers like MAC address, UUID, or other proprietary serial numbers. So we need some alternative of EPC or reasonable conversion or mapping scheme to utilize ONS. In this platform, we developed identification and deployment scheme of BLE Generic Attribute Profile (GATT). The GATT profile is predefined generic profile containing specification of certain application or behavior that can be used for structured communication with BLE devices. Each GATT profile represents functional class of device and can be distinguished with assigned UUID, so we can utilize this for querying ONS to retrieve device information except for some unique information such as serial number of product name. As illustrated in Figure 5b, the user can query with GATT profile service id and retrieve corresponding compatible driver to utilize functionalities of the BLE device.

V. Implementation

We have implemented the prototype of IoT-MAP platform on the top of Android. The platform runs as a service on the Android. The IoT-MAP platform is designed to run on any operating system which supports JVM or compatible engine (e.g. Dalvik), and we implement this on Android to verify its capability.

A. Bridging OSGi Framework and Android Dalvik VM

The interfaces abstracting various functionalities of smart things should be used as a bridge between OSGi Framework and Android application. When the developer asks service object to embedded OSGi Framework, it returns Java *Object* type object. But the execution environment

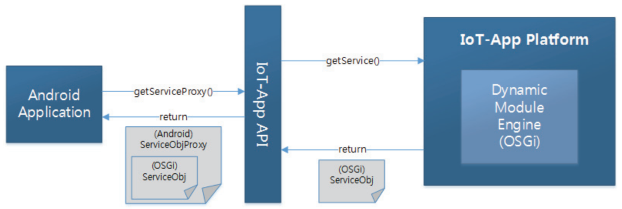


Figure 6: Dynamic Proxy of OSGi Service Object to be used in Android Environment

of OSGi Framework and Android Dalvik VM are isolated with different classloader, so *ClassCastException* is thrown when the developer just try to cast the object from one to another even though same interface is used to cast. This problem that makes two environment cannot be truly integrated is happened not only on Android/OSGi but also on Tomcat/OSGi which has embedded OSGi Framework in JVM. Usually this problem is conquered by registering interfaces to use before execution of OSGi Framework, but it is not an available option in IoT-MAP Platform because of its dynamic aspect. Another option is to use libraries which can modify the Java bytecode dynamically and cast it, but those libraries cannot be used in Android Dalvik VM. So in Dynamic Execution Engine of IoT-MAP Platform, we utilized *Reflection* and *Proxy* to overcome this problem. As illustrated in Figure 6, the platform encapsulates the service object with dynamic proxy of Java API with the same interface. With this improvement, the platform can successfully integrate OSGi Framework and Android applications.

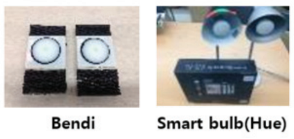
VI. Demo Application and Evaluation

A. Bendi Call Scenario

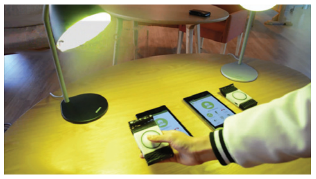
This scenario shows integration of multiple smart things and application developer's business logic. As shown in Figure 7c, the Philips Hue and the tactile interaction device Bendi[14] are integrated in single smartphone to provide composed interactive communication call service. To demonstrate this scenario, we developed simple VoIP call application as an IoT-App. It enables users to make a call to each other, and it also provides additional tactile and visual interaction over just voice. As shown in Figure 7, each device discover, deploy, and integrate services provided by Hue and Bendi, then make them send their input over VoIP SMS service. These additional function are implemented in drastically reduced line of code, because all implementation details for discovery, connection, and controls are delegated to the platform.

B. Versatile App Scenario

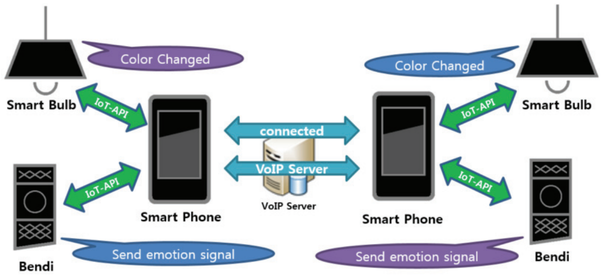
Versatile App[15] is implemented to provide more freedom to user for selecting and composing things around the user's smartphone. The GUI to integrate smart things is implemented based on the open-source service composition



(a) Devices on Scenario



(b) Service Launching



(c) Scenario Outline

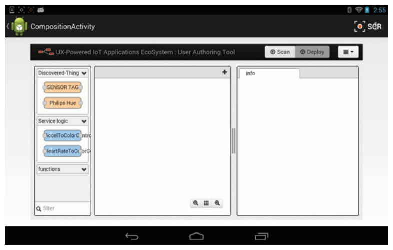
Figure 7: Bendi Call Scenario

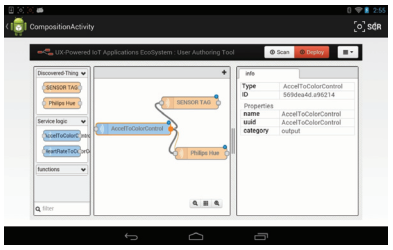
framework Node-RED[12]. In Versatile App, the business logic to integrated smart things is provided in the form of *Service Logic*. This service logic is smaller atomic logic compared to the business logic of general IoT-App, so user can select on their intention. In this scenario, we used TI SensorTag[13] which embeds acceleration sensor and smart bulb Hue. As illustrated in Figure 8, the scanned devices are service logics are shown in the GUI, user connect the devices and logics with touch, and then the service is launched. The values generated by SensorTag is translated to RGB value by service logic, then it is consumed by Hue bulb, so the composition flickers bulbs when user shakes the SensorTag.

C. Overhead Evaluation of IoT-MAP Platform

To evaluate overhead of IoT-MAP Platform, we used virtual stub smart thing implementation and compared time difference with and without IoT-MAP Platform support. This stub full loads to the CPU of smartphone, and the IoT-App runs this stub from 1 to 16 count. The test is run on Nexus 7 (2013) which has quad-core processor and the result is shown in Figure 9.

In the graph, *Android* refers just running stub code in Android Activity, *OSGi Service* refers the stub is directly executed in OSGi Framework, and *IoT-App* refers the stub is loaded and run on IoT-MAP Platform. The exact difference of overhead can be calculated with the *OSGi Service* and *IoT-App*. There is no significant difference when one or two stubs are used, but when more are used, about 5% of overhead is measured. This seems that the thread of IoT-App makes small overhead when 4 or more stub threads are already running. This result is critical condition that the thing lays full load to smartphone, so the actual overhead in normal use will be not significant. The *Android* showed


(a) Smart Thing Discovery


(b) Connecting Logics with Touch Input

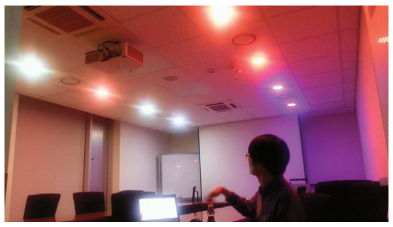

(c) Service Launching

Figure 8: Versatile App Execution Procedure

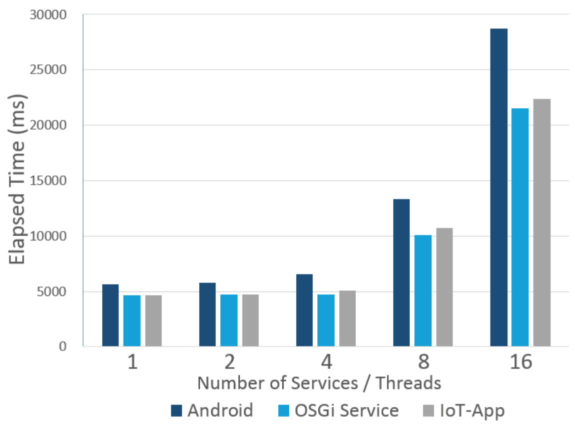


Figure 9: Overhead Evaluation Graph of IoT-MAP Platform

lower performance than others, and it shows OSGi Framework itself can run the stub more efficiently than that run from Android activity thread.

VII. Related Work

Combining available information and services from multiple sources to compose a new service is already popular in World Wide Web technology under the name of Web Mashup. Efforts to apply this concept to smart things resulted studies such as mashup within the Web of Things[4] or within various heterogeneous devices. Also, many scientific studies(e.g. Cilia[7] or Dynamo[8]) and open-source projects(e.g. Eclipse IoT[10] or OpenHAB[9]) are working on how to compose services of smart things and provide new values to users. However, most of these studies and projects usually assume the existence of a certain home server or gateway agent to manage underlying devices.

IoT Mashup as a Service [6] proposed a generalized cloud-based service model which consists of software model, thing model, and computing resource model, and it designed a cloud-based architecture for end-users to create a composed service by assembling software components of application logics and proxies of heterogeneous devices. While IoTMaas targeted cloud-based service model and architecture, our work focused on mobile pervasive environment, so it has a chance to provide great user experience by composing personal and surrounding devices dynamically.

Ambient Dynamix[16] provide a service composition functionality for mobile applications using smart things. It has advanced context sensing capabilities to support community based composition function. However, It only could discover registered devices on Dynamix plug-in server for discovery while IoT-MAP could discover surrounding devices in general way with open-source based well-established name server Oliot-ONS, and it performs ID resolution procedure to retrieve the service list from object name service. The object name service follows DNS hierarchical structure and distributed over the world. So by using the ONS infrastructure, our external repository systems are highly scalable.

VIII. Conclusion

In this paper, we introduced smartphone-centric smart thing service composition platform IoT-MAP Platform that dynamically discover devices, deploy drivers, and provide them in uniform interface to IoT-App. For future work, the method to derive abstracted general interface should be studied with existing profile-based protocols to abstract most of smart things available in market. And also, because the abstracted interface cannot cover all functionalities of each things, platform should provide generalized interface to handle GUI dialog for additional interaction between user and the driver of smart thing.

Acknowledgment

This work was supported by Institute for Information and communications Technology Promotion(IITP) grant funded by the Korea government(MSIP)(No. 10041313, UX-oriented Mobile SW Platform)

References

[1] Traub, Ken, et al. *The GS1 EPCglobal Architecture Framework*, GS1, 2014.
[2] Dean, Kevin, et al. *GS1 Object Name Service (ONS)*, GS1, 2013.
[3] Murugesan, San. *Understanding Web 2.0*, IT Professional, 2007, 9.4: 34-41.

[4] Guinard, Dominique, et al. *Towards physical mashups in the web of things*, In: Networked Sensing Systems (INSS), 2009 Sixth International Conference on. IEEE, 2009. p. 1-4.

[5] Bandyopadhyay, Debasis; Sen, Jaydip. *Internet of things: Applications and challenges in technology and standardization*, Wireless Personal Communications, 2011, 58.1: 49-69.

[6] Im, Janggwan; Kim, Seonghoon; Kim, Daeyoung. *IoT Mashup as a Service: Cloud-Based Mashup Service for the Internet of Things*, In: Services Computing (SCC), 2013 IEEE International Conference on. IEEE, 2013. p. 462-469.

[7] Lalanda, Philippe; Escoffier, Clment; Hamon, Catherine. *Cilia: An autonomic service bus for pervasive environments*, In: Services Computing (SCC), 2014 IEEE International Conference on. IEEE, 2014. p. 488-495.

[8] Avouac, P.-A.; Lalanda, Philippe; Nigay, Laurence. *Adaptable multimodal interfaces in pervasive environments*, In: Consumer Communications and Networking Conference (CCNC), 2012 IEEE. IEEE, 2012. p. 544-548.

[9] OpenHAB. Open HAB site.
http://www.openhab.org/index.html

[10] Eclipse IoT. Eclipse IoT site.
http://iot.eclipse.org/index.html

[11] OSGi Alliance, OSGi site
http://www.osgi.org/

[12] IBM Emerging Technology, Node-RED site
http://nodered.org/

[13] Texas Instrument, CC2541 SensorTag Development Kit site
http://www.ti.com/tool/cc2541dk-sensor

[14] Park, Young-Woo; Park, Joohee; Nam, Tek-Jin. *Bendi: Shape-Changing Mobile Device for a Tactile-Visual Phone Conversation*, In: Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems. ACM, 2015. p. 181-181.

[15] Sungpil Woo, Sehyeon Heo, Janggwan Im, and Daeyoung Kim. *Versatile Internet of Things Application on Mobile Dynamic Service Composition Framework*, In: The 4th International Conference on Internet of Things (IoT 2014), MIT, USA, Oct. 4-8, 2014

[16] Darren Carlson, Andreas Schrader *Dynamix: An Open Plug-and-Play Context Framework for Android*, In: The 3rd International Conference on Internet of Things (IoT 2012)

[17] OMG Architecture Board ORMSC. Model driven architecture (MDA). OMG document number ormsc/2001-07-01, available from www.omg.org, July 2001