The GS1 code based Web of Things Service Architecture with Healthcare Scenario

Jongseok Choi, Minkeun Ha, Nam K Giang, Wondeuk Yoon, Kiwoong Kwon, and Daeyeong Kim
Department of Computer Science
Korea Advanced Institute of Science and Technology (KAIST)
Email: {jongseokchoi, minkeun.ha, zang, wdyoon, kiwoong, kimd}@kaist.ac.kr

Abstract—In the Web of Things (WoT), there were many efforts to integrate physical things to the Web. However, in the WoT web app implementation, there are many issues such as resource constraints and connectivity of physical things, which should be handled. We explain how these problems can be resolved with our GS1 code based WoT architecture and show the feasibility of the architecture with the integration of our IPv6 over Bluetooth Low Energy technology, the app server dealing with resource constraints of physical things, healthcare web app, and the end user device, Google Glass.

I. INTRODUCTION
The Web of Things (WoT), which integrates trillions of things to the Web, expands the current Web eco-systems to provide innovative WoT services with rich user experience and user-friendly interfaces. However, the major problem here is the fact that most of physical things are equipped with limited resource and constrained networking interfaces. To overcome this problem, our previous work [1] suggested a thing browsing architecture with introducing an app server which deals with static web contents such as multimedia, CSS, and Javascript instead of physical things. However, its redirection approach causes high traffic overhead and violates Same Origin Policy (SOP). To relieve these issues, we suggested a new cross domain communication method in [2]. Although this method avoids SOP problem, physical things still should deal with the high Internet traffic.

To solve this problem, we propose a GS1 code based Web of Things architecture which minimizes the Internet traffic towards physical things except a vital communication to obtain real-time sensor data. In this architecture, an app server first serves HTTP requests of users and responds web contents with a simple code to obtain real-time sensor data using SCoAP [3]. The user web browser then queries sensor data to the physical things and creates a complete web service by merging them. With this architecture, web service developers are able to implement and deploy new WoT services by simply locating the service contents into the app server without modifying physical things. However, this approach requires identification method for web contents stored in app server, since the app server is a WoT service entrance which gives direction to browse the things. For this, we suggest GS1 code based domain name service to identify the required web service from the service requests. By minimizing interaction with physical things located in low-rate networks which causes performance degradation, this approach reduces the web service response time.

For our demonstration, we implement a patient-centric healthcare system as a WoT service with IPv6 over Bluetooth Low Energy (6BLE) healthcare devices [6]. Our 6BLE devices contain all the key features of the draft [5] and additional functionalities such as fragmentation & reassembly and so on. This technology provides interoperable interaction method between the legacy Internet and 6BLE networks via patient’s smartphone which acts as an edge router. With the help of the app server, the healthcare web app that enables a doctor to browse his patients is implemented. By experiencing the healthcare web app on a user device, Google Glass, finally, we show the feasibility of our GS1 code based WoT architecture.

This paper is organized into four sections. In section II, we explain our approach. Next in section III, we present the implementation of the system with our healthcare service. We conclude in section V.

II. THE GS1 CODE BASED WoT ARCHITECTURE
Since physical things are usually under constrained environment, the communication with the things is a bottleneck of the service in the WoT web service. To resolve this bottleneck, the app server handles the traffic except the essential communication of real-time sensor data acquisition with the things. By providing the development environment to deal with constraints of the things, the app server also helps web developers easily implement their web app. Especially, the APIs for connectivity protocols such as CoAP lessen concerns of developers about communication overhead under constrained environment. At the same time, the web service, dev.example.com, enables developers to deploy their web apps on the app server. In this approach, however, we can not identify the requested web app from the traffic, since there is no method to decide the web app from the traffic. For this issue, we make use of the domain name. The app server generates a GS1 code based domain name per thing/mashup service formed in service-serial.service-class.app.example.com. For example, the domain name of a heart-rate sensor can be generated based on its GS1 code as shown in the Fig. 1.

Using the domain name, data flow of our system is described as Fig. 2. When a user requires a WoT service, www.myhealth.com, he can type it on his web browser. To resolve the IP address of www.myhealth.com, the web browser sends a request to a DNS server. Since www.myhealth.com is registered as a CNAME record as ‘www.myhealth.com. IN CNAME 1234567890.006141999996.app.example.com.’, the DNS server sends ’1234567890.006141999996.app.example.com’ as a response. When it reaches to the web browser, the web browser requests the IP address of the acquired domain again and finally obtains the IP address of the web service. From
the revealed IP address, the web browser sends a HTTP request for web pages and contents to the app server. After the app server respond to the browser, the browser processes received web pages and can request sensor data to a physical thing, which is identified from the GS1 code based domain name, '1234567890.00614141999996.app.example.com'. If the thing is running a CoAP server for data communication, the browser can send a CoAP packet through the our previous work, SCoAP [3], a technique with HTTP-CoAP proxies using HTML5 WebSocket and a JavaScript creating and parsing CoAP packet from HTTP packet. In this mechanism, our app server makes it possible for physical things to save their resource.

In addition, GS1 code based additional information of physical things from Object Name Service (ONS), a mechanism that leverages Domain Name System to discover information about a product and related services from the Electronic Product Code, and Discovery Service (DS), a service to find and obtain all relevant visibility data defined by GS1, web developers can add more functionalities to their web app.

### III. Implementation

For our demonstration, we use a heart-rate sensor, Mio Link and e-Health sensor platform for electrocardiogram (ECG), electromyograph (EMG), galvanic skin response (GSR), and thermo and Google Glass for the user device shown in the Fig. 3. We created an Android application receiving heart-rate data from the Mio Link via the Bluetooth Low Energy (BLE) and passing the data. For sensor data transmission from the e-Health sensor platform, we use our 6BLE technology for network layer protocol. Our 6BLE platform is built on our SNAIL [4] platform. It implements not only all the features of the draft [5] from IETF 6Lo working group, but also fragmentation & reassembly, unit and multicast address mapping, and routing in local network which are not defined in the draft yet. It provides interoperability between IPv6 and BLE by way of smartphones as gateway. With this technique, a smart phone does not need to run many Bluetooth applications pulling and pushing data. Over 6BLE, the developers can handle E-health platform health signals through the CoAP.

Our healthcare web app is implemented under the development environment of the app server and then deployed there. By resolving the network connectivity of healthcare sensors through 6BLE and CoAP, we deliver our healthcare service to the end user device such as the Google Glass which contains its own web browser. Finally, with the integration of our 6BLE, the app server, healthcare web app, we can show the feasibility of our architecture through the WoT healthcare service on the user device, Google Glass.

### IV. Demonstration

Our healthcare web app helps a patient obtain simple medical diagnosis service from his doctor by attaching healthcare sensors to his body. When the patient requests for doctor to check his health signals, doctor can check on the web browser with his desktop or Google Glass. This service helps the doctor quickly browse his patients and check their health signals. It can draw rapid the decision of patient’s condition who is in an ambulance or an emergency room.

### V. Conclusion

The app server enhanced resource usability of physical things in constrained environment and offers development conveniences to developers. With the integration of this app server, the 6BLE technology, healthcare web app and the Google Glass, we showed the feasibility of our WoT architecture. From this benefits and validations, we expect our platforms will give more capabilities to developers to expose surroundings on the Web as user-friendly services. In the future, we will add a function to easily add business logics of stakeholders to our system. Also we will make this system more distributed with the concept of other GS1 platforms.

### VI. Acknowledgements

This research was supported by the International Research 
& Development Program of the National Research Foundation(NRF) funded by the Ministry of Science, ICT & Future Planning(Grant number: 2013064540) and also supported by the Doctor M program of KAIST (Korea Advanced Institute of Science and Technology).

### REFERENCES


