Efficiently Managing Collected from External Wireless Sensors on Smart Devices Using a Sensor Virtualization Framework

Byung-Bog Lee, SangGi Hong, Kyeseon Lee, Naesoo Kim and JeongGil Ko
Electronics and Telecommunications Research Institute (ETRI)

Abstract

By interacting with external wireless sensors, smartphones can gather high-fidelity data on the surrounding environment to develop various environment-aware, personalized applications. In this work we introduce the sensor virtualization module (SVM), which virtualizes external sensors so that smartphone applications can easily utilize a large number of external sensing resources. Implemented on the Android platform, our SVM simplifies the management of external sensors by abstracting them as virtual sensors to provide the capability of resolving conflicting data requests from multiple applications and also allowing sensor data fusion for data from different sensors to create new customized sensors elements. We envision our SVM to open the possibilities of designing novel personalized smartphone applications.

Ⅰ. Introduction

Low-power sensors can be used in various environments so that they can effectively collect various characteristics of the surrounding environment to form useful applications. As a widely distributed mobile computing platform, smartphones possess a number of on-board sensors (e.g., GPS, accelerometer, light sensor, etc.) for various applications to utilize and enrich their quality. Nevertheless, the capability to add additional sensors to a smartphone can further enable new applications and provide useful services for its users. While extending a smartphone's sensing capability (internally) is difficult, fortunately, by using Bluetooth, WiFi, NFC, or Zigbee (with a IEEE 802.15.4 radio attached to a smartphone via USB), external sensors with wireless connectivity can provide high-resolution data regarding the surrounding environment and its inhabitants. Such data can enable various novel smartphone applications that take the accurate local environmental factors in consideration. Examples of these applications include personal healthcare, private living environment control, and more generally, social Internet of Things (IoT) systems [1]. Such a system architecture where external sensors communicate directly with smartphones (e.g., without Internet connectivity) can allow privacy sensitive sensor resources to associate on a per-smartphone basis and securely report their data.

Considering that the same set of external sensing modules can be utilized in multiple smartphone applications simultaneously, these applications can benefit heavily from a software module that manages the usage of the shared resources (e.g., sensors, module parameters, etc.) [2]. We would expect this "sensor controlling module" to be implemented beneath the applications and expose interfaces for applications to easily discover, access and configure locally present external sensors.

In this work we introduce the design and architecture of a sensor virtualization module (SVM), a software component that manages the profiles and data from external wireless sensors. The proposed SVM provides a set of APIs for applications to access external sensors, resolves conflicts among different sensor data requests, and allows the design a new virtual sensors by performing data fusion using data from existing sensors.

This article is structured as follows. In Section II we discuss about the system architecture with sensor network and smart device interaction, and Section III introduces our SVM framework. We introduce a set of applications developed using our proposed SVM framework in Section IV and conclude the paper with directions for future work in Section V.

Ⅱ. System Architecture

In this work we exploit the smartphone as a device that uses the data from the sensor data to enable new applications on the smartphone. In other words, the smartphone acts as a data-consuming terminal that actively collects data from various external sensors to use them in the smartphone's applications. Furthermore, this data can be configured to be forwarded to a back-end server after on-smartphone data processing. In Figure 1 we draw a diagram of how such a system operates in various real world scenarios.

We acknowledge that smartphones can interact with real-world sensors in two major ways. First, is by acting as a gateway that simply forwards the data from external sensors to a server on the web and the second is the data consumer approach we introduce in this work. While currently, the first approach is widely utilized, we foresee that this approach has many limitations which are mostly introduced from the lack or the concerns of data privacy and security. In other words, many personalized sensors carry information that user may or may not want to share with the users on the Internet. For these data, it is meaningless, and even dangerous, to upload to a web and re-download the information for on-smartphone data processing. By exploiting the smartphone as a final data consumer, such privacy sensitive applications can be widely implemented without the user, and the developers worrying (too much) about preserving the privacy of the data.

The challenge behind enabling such system architecture lies in the data management of the sensory data collected from external sensors. The rest of this work introduces our approach to effectively manage these data in a smartphone, or any other smart device connected with external sensors.

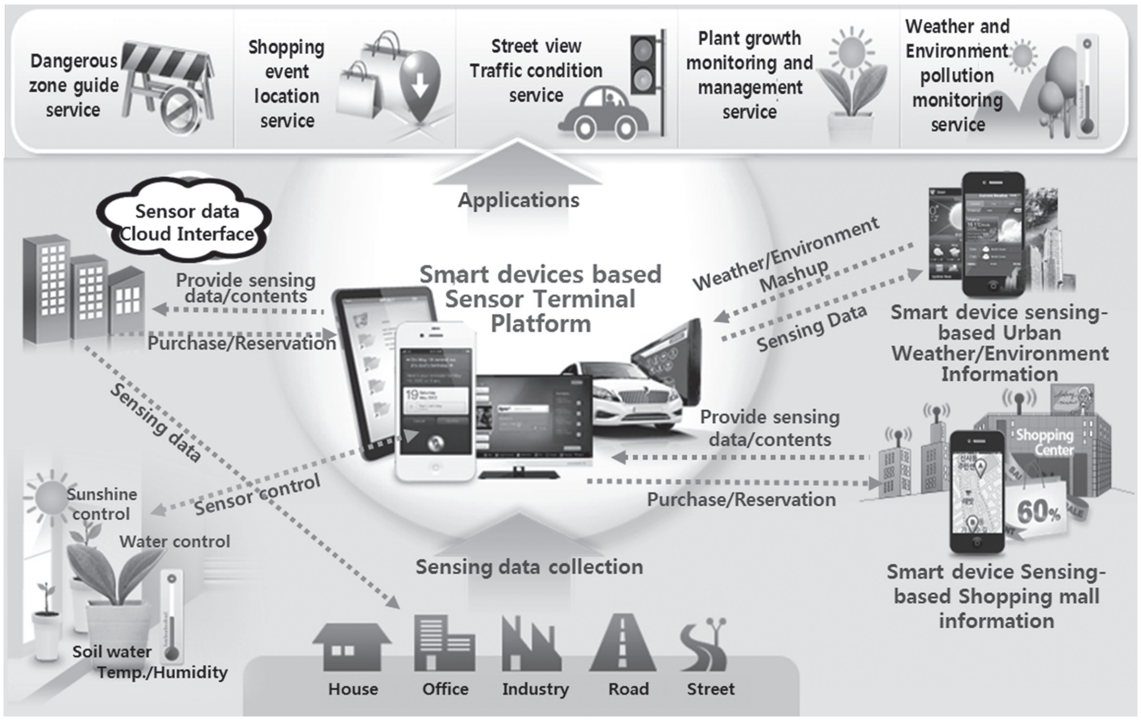
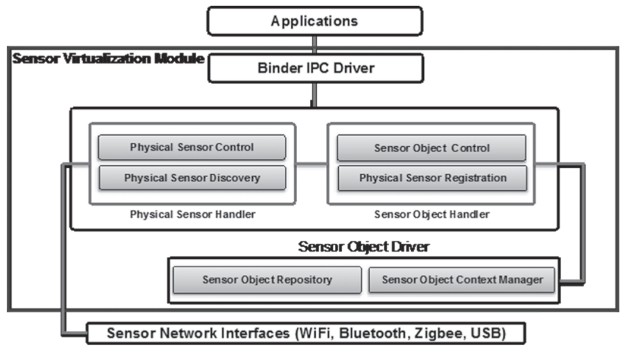

Dangerous zone guide service
Shopping event location service
Street view Traffic condition service
Plant growth monitoring and management service
Weather and Environment pollution monitoring service
Applications
Sensor data Cloud Interface
Provide sensing data/contents
Purchase/Reservation
Smart devices based Sensor Terminal Platform
Weather/Environment Mashup
Sensing Data
Smart device sensing-based Urban Weather/Environment Information
Sensing data
Sensor control
Sunshine control
Water control
Provide sensing data/contents
Purchase/Reservation
Shopping Center
60%
Sensing data collection
Smart device Sensing-based Shopping mall information
Soil water Temp./Humidity
House
Office
Industry
Road
Street

Figure 1. System architecture of various applications enabled using smartphone-centered external sensor field interaction.

Ⅲ. The Sensor Virtualization Module


Applications
Sensor Virtualization Module
Binder IPC Driver
Physical Sensor Control
Physical Sensor Discovery
Physical Sensor Handler
Sensor Object Control
Physical Sensor Registration
Sensor Object Handler
Sensor Object Driver
Sensor Object Repository
Sensor Object Context Manager
Sensor Network Interfaces (WiFi, Bluetooth, Zigbee, USB)
Figure 2. Our proposed sensor virtualization module (SVM) architecture. The SVM provides applications with interfaces and interacts with various network interfaces to control external sensors.

Our implementation of the sensor virtualization module (SVM) on the Android platform, illustrated in Figure 2, performs two major tasks. First, the SVM virtualizes external physical sensors so that applications can easily access the sensing information that they offer. For this purpose, our SVM exposes a set of interfaces that applications can use to discover external sensors, control parameters such as re- porting and radio duty cycling intervals (Physical Sensor Handler in Figure 2), and store/retrieve sensor data to/from an internally managed repository so that data can be shared among different applications (Sensor Object Handler and Sensor Object Driver in Figure 3).

The second major role of the SVM is to resolve conflicts caused by sensor data requests from different applications (Binder IPC in Figure 2). When multiple applications request for data from the same sensor with conflicting characteristics (e.g., different reporting intervals requested from two applications), the SVM can computes the best option to satisfy (all of) the incoming requests. Without this conflict resolution feature at the SVM, external sensor devices would be expected to perform this functionality. Nevertheless, such application-level algorithms are not enforced by any standards; thus, smartphone applications cannot assure that conflict resolution from different requests would be possible at external sensor devices themselves.

Since our current version of the SVM is implemented for the Android application development environment, we note that the SVM operates as a background process. While managing both on-board and external sensors, most components in the SVM are not executed until receiving a request from the application in order to minimize the added current draw that an additional hardware controlling module may introduce.

Managing external sensors using our proposed SVM provides three distinct benefits. First, since the SVM restricts all data to/from external sensors to pass through a single data path, it helps simplify the development process of applications by providing a common API to access any available sensor in the field. With the standards designed for sensor networks [3], new sensors can be quickly discovered and their data can be reported to requesting applications through the virtualization process. Second, the sensor profiles and data that our SVM exposes can be a way of sharing data among different applications. When multiple applications request for data from the same sensor, the SVM, by utilizing the sensor repository, can reduce the number of interactions be- tween the sensors and the smartphone to conserve power at both ends. Finally, as Figure 2 shows, the SVM allows users to "build" custom virtual sensors. Specifically, using existing data in the data repository, users can "mash-up" or perform data fusion on multiple sensor readings to form a new virtual sensor, which is updated along with the original sensor readings (or with a user defined interval). For example, an application can form and register a discomfort-index sensor using

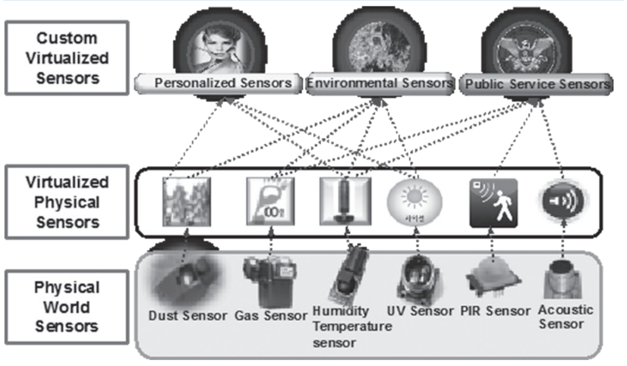


Figure 3. Diagram of the custom virtualized sensor generation process using data from physical sensors. The SVM exposes data from physical sensors in a virtualized format and also allows the combination (e.g., mashup or fusion) of multiple sensors.

the combination of local humidity and temperature readings. This new custom virtual sensor, registered in the repository, can be configured as an open sensor to be used by other applications as well. We will later discuss in detail on the creation process of these sensors in the last part of Section Ⅳ.

Ⅳ. Applications and Use Cases

Using our proposed SVM component, we designed a set of applications and present them in this section. Furthermore, we showcase a number of development scenarios that can benefit from the use of our SVM by easing the development effort for applications that require the utilization of external sensor components with various wireless (or wired) networking capabilities.

Application 1 – Smart TV-based Robot Vacuum Cleaner Management: As one of our initial prototype application implementations, in collaboration with our industrial research partners, we have designed a simple application using the SVM that locally controls widely used robot vacuum cleaners using the remote control system at a smart TV as illustrated in Figure 4.

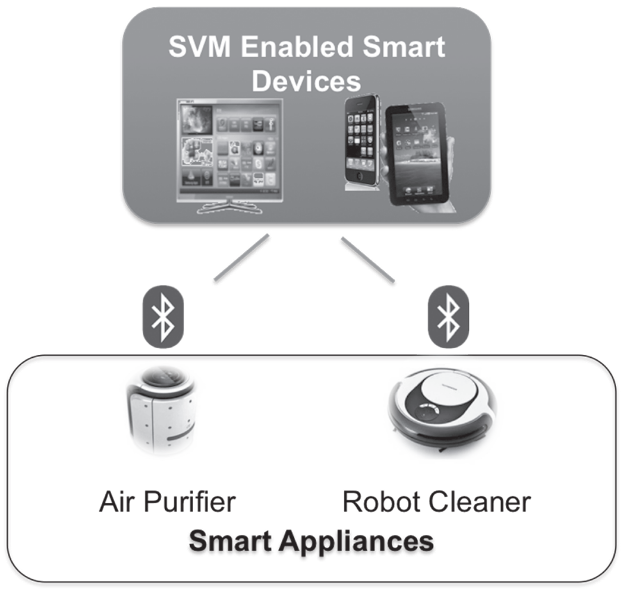


Figure 4. System architecture for our smart appliance (smart house cleaning) application prototype.

When using the SVM, the application program first starts by issuing a command to the SVM to search for a robot cleaner in its communication range. The application first configures the SVM so that it starts searching with an expected radio module of the robot cleaner (in our case a Bluetooth radio module) and the SVM identifies a potential hardware component that can be used. This search result is reported to the application from the SVM and the application issues a user input query to select one of the (many) devices that were found. The SVM does not restrict the message format between the robot cleaner (or any other external sensing device) and the interacting application. The only mandatory requirement to interact with the SVM is configuring the resource discovery format so that the SVM knows what the discovered sensing/actuation resource is. Essentially, this is the only step that an application needs to take to connect to an external sensing component. This makes the development process simple and potentially encourages application developers to easily design applications that connect to external sensors or actuators to gain additional information on the environment and/or control parameters.

Application 2 -- Smartphone-based At-home Environmental Sensor Management: One of the most intuitive applications that one can think of when drawing the picture of the SVM discussed above and its capabilities, is easily accessing wireless sensor network nodes' data from a smartphone. Traditionally, these sensor nodes had their own Internet connected gateway devices and smartphone applications would access the sensing data via a web interface. Nevertheless, by using the APIs provided by the SVM, accessing such sensor information directly becomes a simple process.

Similar to our application scenario discussed above for the robot cleaner application, connecting application-specific sensors for different purposes becomes a simple process when utilizing the SVM. As mentioned earlier, applications will simply issue commands that describe the sensor that the application is looking for (e.g., type

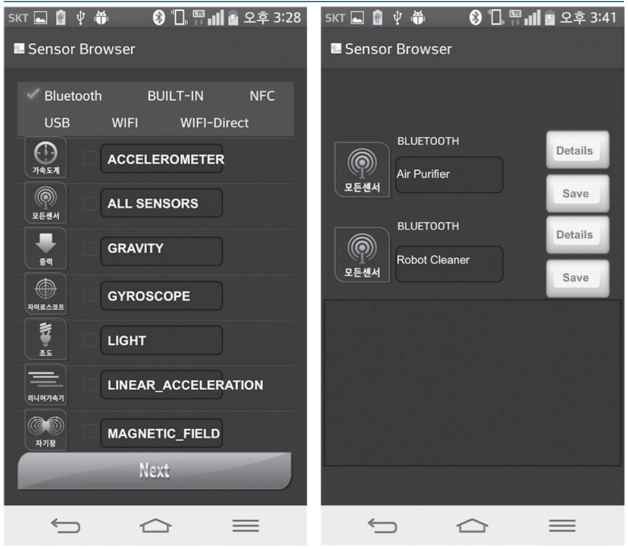


Figure 5. Screenshot taken from our sensor browser application which allows easy access and repository registration for various external sensors.

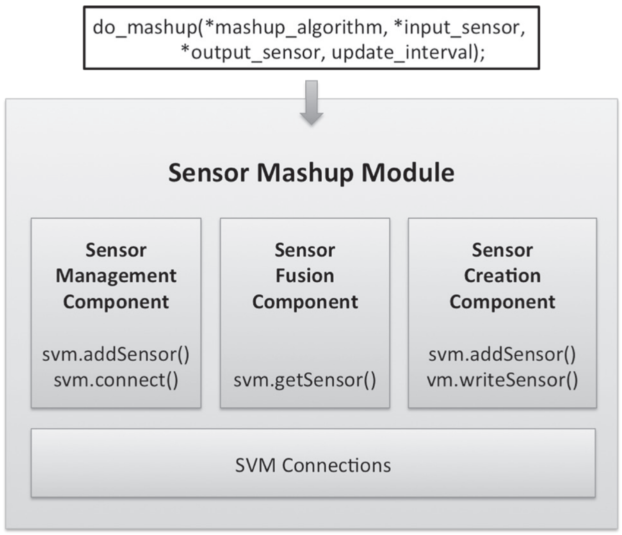


Figure 6. Function diagram of our sensor mashup framework implemented for the SVM.

of sensor, connection type, etc.) and select the sensor to connect to. The SVM will simply collect the sensing data in its repository and allow authorized applications to access the data freely. Surprisingly, we emphasize that such features are achieved using a very simple set of APIs. For example, to check whether a sensor data is already tracked in the SVM sensor repository, the application simply issues a checkInRepo(SENSORID) command. To add a new sensor to the repository, the addSensor(SENSORID) and connect(SENSORID) commands help add a new sensor in the environment to the repository. As a result of these simple set of APIs from the SVM, an application can neglect the process of connecting to a sensor using the native networking APIs and only focus on designing the operations that use the sensor information to form the information required at the applications. Our sensor browser application (see Figure 5) exposes these APIs to allow easy discovery and registration of various external sensors to the SVM's sensor data repository.

Application 3 -- Virtual Sensor Creation and Sensor Data Fusion with SVM: Lastly, we demonstrate the process of merging different sensor information for form a new set of virtual sensors that are targeted for application specific purposes. This is a unique feature of our proposed SVM and has the potential to create a diverse set of applications, which utilize sensor data fusion in an easier fashion. Our illustrations of the virtual sensor

The development process for a new virtual sensor is fairly simple when using the SVM's APIs. Essentially, as Figure 6 shows, we have designed a sensor mashup framework to lie on top of the SVM. This framework exposes a simple, single API that allows the application layer to provide the set of input sensors, target output sensor, target mashup function, and the virtual sensor update interval. Specifically, the single command that the sensor mashup framework exposes to the applications is described as the following.

```
do_mashup(*mashup_function,
  *input_sensors,
  *output_sensor,
  update_interval);
```

where *mashup_function is the actual desired fuction of how the set of input sensors (e.g., *input_sensors) are computed to store the result in the sensor ID *output_sensor, which is updated with an interval of update interval. The result of this computation is added to the SVM's sensor repository using the write_sensor(output_sensor) command. With this operation, the application can make new sensors, which are a result of various sensors that already exist in the repository. In the internals, the sensor mashup module (or framework) essentially, uses the APIs provided by the SVM to manage or retrieve data from the repository so that later on, the application can easily access the new virtual (mashed-up) sensor data using the SVM's getSensor() command.

V. Conclusion

In this work we introduce the sensor virtualization module (SVM), which helps smart devices interact with external sensors that collect data from the physical environment. The goal of our SVM is to provide an easily accessible development environment to application developers that target to design applications that use sensing data from wireless low-power sensors. By providing a set of API that applications can access to manage external sensing and actuation components, our SVM simplifies the application development process dramatically. Given that the world of cyber physical systems would effectively require smart interaction between low-power embedded platforms in the environment with smart devices that interact with humans, we believe that our SVM provides an easy accessible development framework that catalyzes the realization of various cyber physical system designs.

References

[1] Luigi Atzori, Antonio Iera, Giacomo Morabito, and Michele Nitti. The social internet of things: When social networks meet the internet of things: Concept, architecture and network characterization. Computer Networks, 56(16):3594 – 3608, 2012.

[2] Rohit Chaudhri, Waylon Brunette, Mayank Goel, Rita Sodt, Jaylen VanOrden, Michael Falcone, and Gaetano Borriello. Open data kit sensors: mobile data collection with wired and wireless sensors. In ACM Symposium on Computing for Development, 2012.

[3] JeongGil Ko, Stephen Dawson-Haggerty, David E. Culler, Jonthan W. Hui, Philip Levis, and Andreas Terzis. Connecting Low-power and Lossy Networks to the Internet. Communications Magazine, IEEE, 49(4):96 –101, April 2011.

약 력

이 병 복

1991년 호원대학교 전산학 학사
1993년 전북대학교 전산학 이학석사
2010년 고려대학교 전산학 이학박사
1993년~현재 한국전자통신연구원 (ETRI) 책임연구원
관심분야: 이동통신시스템 응용기술, 사물통신 시스템 설계

홍 상 기

1997년 부산대학교 학사
1999년 부산대학교 석사
2013년 충남대학교 박사
2001년~현재 한국전자통신연구원 (ETRI) 선임연구원
관심분야: Sensor Signal Processing, IoT Collaborative System and Framework

이 계 선

1998년 충남대학교 학사
2000년 충남대학교 석사
2000년~현재 한국전자통신연구원 (ETRI) 선임연구원
관심분야: CPS Application Design, Internet of Things System Design

김 내 수

2007년 한남대학교 박사
1976년~1990년 국방과학연구소 근무
1990년~2012년한국전자통신연구원 (ETRI) 팀장
2013년~현재 한국전자통신연구원 (ETRI) 연구책임자/책임연구원
관심분야: RFID/USN, Internet of Things(IoT), 위성통신

고 정 길

2007년 고려대학교 학사
2009년 Johns Hopkins Univ. 석사
2012년 Johns Hopkins Univ. 박사
2012년~현재 한국전자통신연구원 (ETRI) 연구원
관심분야: CPS Application Design, Internet of Things System Design