Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2015, Article ID 730762, 10 pages
http://dx.doi.org/10.1155/2015/730762

Research Article

Sensor Virtualization Module: Virtualizing IoT Devices on Mobile Smartphones for Effective Sensor Data Management

JeongGil Ko,[1] Byung-Bog Lee,[2] Kyesun Lee,[2] Sang Gi Hong,[2] Naesoo Kim,[2] and Jeongyeup Paek[3]

[1]*Department of Software Convergence Technology, Ajou University, Suwon 16499, Republic of Korea*
[2]*Electronics and Telecommunications Research Institute (ETRI), Daejeon 34129, Republic of Korea*
[3]*School of Computer Science and Engineering, Chung-Ang University, Seoul 06974, Republic of Korea*

Correspondence should be addressed to Jeongyeup Paek; jpaek@cau.ac.kr

Received 15 May 2015; Accepted 3 August 2015

Academic Editor: Sung Won Kim

The vision of the *Internet of Things (IoT)* is coming closer to reality as a large number of embedded devices are introduced to our everyday environments. For many commercial IoT devices, ubiquitously connected mobile platforms can provide global connectivity and enable various applications. Nevertheless, the types of IoT resource-utilizing applications are still limited due to the traditional stovepipe software architecture, where the vendors provide supporting software on an end-to-end basis. This paper tries to address this issue by introducing the *Sensor Virtualization Module (SVM)*, which provides a software abstraction for external IoT objects and allows applications to easily utilize various IoT resources through open APIs. We implement the SVM on both Android and iOS and show that the SVM architecture can lead to easy development of applications. We envision that this simplification in application development will catalyze the development of various IoT services.

1. Introduction

Low-power embedded sensor networking platforms will soon be deployed for various application purposes in our everyday environments. On large scale, applications such as smart power-grids [1–3], smart city management [4], home and building automation [5, 6], wireless sensor networks [7, 8], and the newly proposed concept of Industry-4.0 [9, 10] will quickly increase the number of embedded computing platforms dramatically. Embedded computing platforms for these applications will be deployed in a way such that the nodes or networks that consist of these systems will have a way to interconnect themselves with the larger Internet architecture [11, 12]. With such advances in embedded devices and networks, increase of computational power, and the ability to interconnect with other devices, the concept of *Internet of Things (IoT)* has emerged. These IoT devices will mostly utilize standard protocols and mechanisms to communicate with the Internet and transport their data to consumers and services in the cloud [13]. To access these IoT devices, a service providing component on the cloud will advertise the resources of these devices for applications to discover and utilize. The ability to connect, communicate with, and remotely manage millions of networked, automated devices via the Internet has opened the potentials for the development of various applications that impact our everyday lives.

Nevertheless, the concept of IoT means more than just devices connecting to the global Internet. There will be another class of IoT devices with *some* radio connectivity that can connect to nearby devices for local communication but not to the global Internet by itself. Not only will miniature sized IoT devices be deployed as network-scale services but we will also enjoy many personalized IoT services which involve only a small number of devices with such limited connectivity. As numerous interconnected devices that can sense and control are embedded closely to our living environments, we envision that there will be a larger number of these "connection-limited" devices that allow more local and personalized applications to be designed. Examples of such

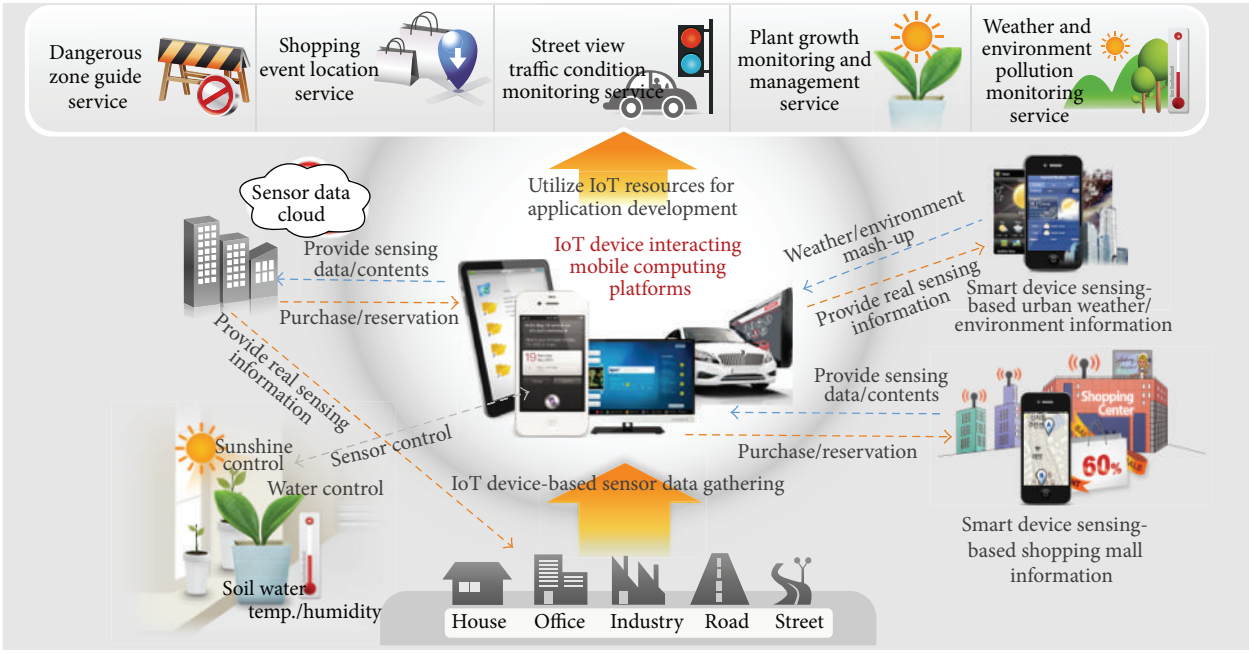


FIGURE 1: Example applications where IoT platforms are interacting with mobile computing platforms.

include IoT devices for personal health exercise management and home monitoring and control.

Due to their size, scale, and resource constraints, devices manufactured for these services are unlikely to provide global connectivity by themselves. Luckily we can easily find many ubiquitously connected mobile computing platforms (e.g., smartphones, tablets) [14]. Furthermore, many of the low-resource IoT devices are designed so that they interact with and through these mobile platforms. Recent work by Park et al. shows that the computational power of current-day smartphones is capable of supporting an even more diverse set of wireless standards using cognitive or software radios [15]; thus, we will soon be seeing such mobile platforms interact heavily with sensing systems using various wireless standards. By connecting IoT devices to these mobile computing platforms, we can quickly realize various mobile platform-based IoT applications. And with the mobile computing platforms' global connectivity, we can easily expose IoT resources globally to allow remote platforms and services to utilize local IoT devices. Figure 1 illustrates examples of such application scenarios where individual IoT devices are connected to user-controlled mobile devices and an associated cloud service takes the role of exposing the IoT resources for other services to reuse. These IoT devices exploit the ubiquitous connectivity of a mobile platform to gain the always-on connectivity required to reach other destinations on the Internet.

While being an attractive scenario, however, many of these IoT devices interacting with mobile devices today can only interface with a software stack that is designed by the manufacturer because the software architecture of these devices mostly takes a "stovepipe" approach, where the vendor of the IoT device provides a complete stack of (closed) software implementation to fully exploit their functionalities [16]. In such a software design, it is difficult for various third-party applications to fully utilize the IoT devices since vendor-provided APIs are usually limited. Furthermore, this restricts resource sharing among different applications, not only for applications internal to a single mobile platform, but also for external applications running in the cloud or on remote smartphones. By breaking down the bricks of the software "stovepipe" and by gaining the ability to handle such information in a more uniform manner, we can allow the same set of IoT devices to be shared, used, and managed in a more flexible way. This will increase their usability in various scenarios, and a more diverse set of applications can be developed and distributed.

This paper introduces the *Sensor Virtualization Module (SVM)*, designed to provide an abstraction for accessing, managing, and sharing the data and resources provided by embedded IoT devices. The proposed *SVM* not only provides global connectivity and service exposure to the Internet for IoT devices but also provides a set of open APIs for mobile applications to utilize and provides ways to design an application server on the cloud for mobile computing platforms or Internet-based services to access remote IoT resources. Furthermore, the *SVM* also allows conflict resolution between different IoT resource requests to allow efficient sharing of IoT resources across multiple IoT applications and also provides a feature that allows creating virtual IoT devices using sensor data mash-up. Finally, *SVM* includes software reprogramming capability that supports software update of both the IoT device and also the mobile computing platform to which local IoT devices are connected.

Using a prototype *SVM* implementation, this paper also introduces a number of applications that can benefit from the resource sharing functionality that the *SVM* provides.

The remainder of this paper is structured as follows. In Section 2 we describe the *SVM* architecture and explain the two important features of *SVM*: creating virtual IoT device for sensor data mash-up and software reprogramming. Then in Section 3, we present two application case studies that make use of and benefit from the *SVM* architecture: sensor data management application and the home appliance management application. Finally, we summarize our work and discuss future research directions in Section 5.

2. Architecture

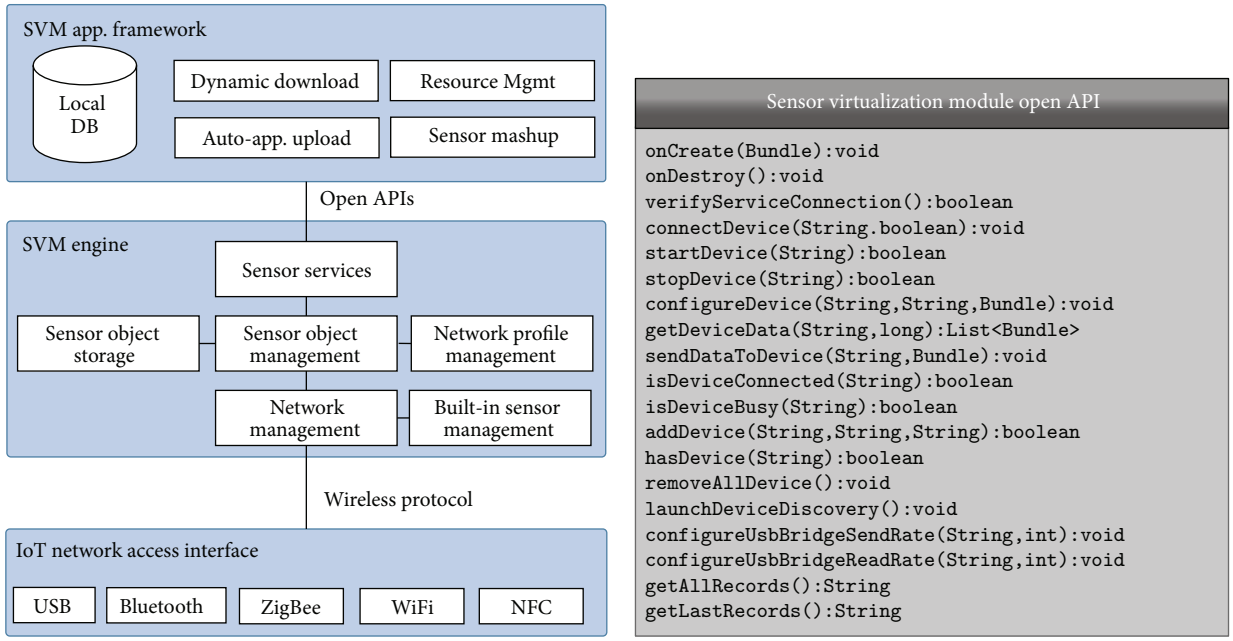
To address the aforementioned challenges, this section introduces the architecture of the *Sensor Virtualization Module (SVM)*, which is designed to provide individual external IoT devices with the connectivity, exposure, and management features required to be effectively utilized over various applications. By external IoT devices, we mean embedded devices such as sensor, actuators, and home appliances with some form of networking and computation capabilities to interconnect and communicate with other local devices, but without direct global connectivity to the Internet and not internal to a mobile device with global connectivity.

2.1. Sensor Virtualization Module. Even in the market today, we see a variety of IoT devices that report their data to services on the Internet using the connectivity of a mobile computing platform. With the proliferation of user-friendly mobile and web applications, this trend will increase. In scenarios where a mobile device takes a significant role for IoT platforms, it is important that the mobile device manages its "reachable" resources efficiently so that the resources provided by IoT devices can be easily accessed, controlled, and shared by various applications.

To address this issue, we design *SVM*, which takes on the role of providing an abstraction for IoT resources that a mobile computing platform can access in its local network. Specifically, *SVM* provides support for applications to access the external IoT resources by first abstracting the network interfaces. For this, upon request from upper-layer applications, the *SVM* engine starts a device discovery phase in which it activates all possible network interfaces to search profiles of IoT devices within reach. Based on the result of this device discovery, the *SVM* engine formulates device object handlers for each identified IoT device. These handlers include information on the name of the device, networking interface, network address, and other device specific profiles. The object handlers are then advertised to the applications, allowing them to select the devices they wish to connect with. As a result of this process, applications can access external IoT devices without knowing how the devices are physically connected. We illustrate the software architecture of our proposed *SVM* in Figure 2(a).

SVM simplifies application development by providing an abstraction of the IoT devices that are present in the local field. In other words, *SVM* makes it look as if the external devices are on-board sensor components such as accelerometer, gyro, GPS, or camera. When applications interconnect with the *SVM* layer, a set of open APIs are provided, as exemplified in Figure 2(b), to the applications for them to interact with the *SVM* layer and to easily access these resources. For example, `launchSensorDiscovery()` API searches for all the external physical sensors that can be connected to the smart device via any of the network access interfaces (e.g., Bluetooth, WiFi, and ZigBee) on the smart device and provides a list of those sensors. `connectDevice(string)` API takes the identifier (e.g., MAC address) as an argument and connects to the external sensor and creates an object within the memory of the smart device. Also, `getDeviceData(string,long):List<bundle>` API takes the sensor identifier and the maximum number of data readings to take as the arguments, reads the sensor data from the external sensors as a list of data items, and inserts them into the internal memory within the sensor data object at the smart devices. (Full API documentation list can be found in the ETRI internal technical report and can be given upon request.) While we expect the application to already understand the format of the incoming data from the external IoT device, an additional data translating module in *SVM* can be used to convert device specific data format to a common XML format. In other words, this allows the users to freely design and utilize their own application layer protocol, such as CoAP. The focus on the *SVM* design was to minimize restrictions to the application system developers while freeing them from the fuss of dealing with low-level communication and networking details in designing an IoT application system. Overall, the use of the *SVM* makes the development process of smart device applications that utilize external IoT devices easier.

Although this process allows easy access to external IoT devices via mobile platform from various applications, the increase in the number of interacting applications can result in conflicts among resource requests. For example, applications *A* and *B* can each ask for data from the same IoT device but ask for it to be retrieved at different time intervals or may potentially request the device to be actuated in different ways. However, since such application level algorithms are not enforced by any standardization body, we cannot assure that these conflicts will be properly processed at the end-device level. As a result, besides managing the connectivity, another major role of the *SVM* engine is to resolve such conflicts caused by multiple requests from different applications. When multiple applications make conflicting requests, *SVM* runs its rule-engine within the `Sensor Object Management` module to select the best option to satisfy most of the incoming requests. The conflict manager of the `Sensor Object Management` operates based on a policy which an administrator user can input at runtime. For example, application *A* may ask for sensor readings every 10 minutes, and a different application *B* may ask for sensor readings every 5 minutes, both from the same external IoT device. In this case, the external sensor can read out sensor samples every 5 minutes but send it to application *A* only for every other reading while sending



(a) *SVM* architecture (b) *SVM* open APIs

Figure 2: *SVM* software architecture and summary of *SVM*'s open APIs.

all readings to application *B*. In this case, the rule-engine is given a policy to take the greatest common factor for meeting the schedule of multiple applications. Also similarly, when multiple applications request different duty-cycle operations on an IoT device for energy management, given the goal of providing a satisfactory QoS for all connected applications, the larger operation duty-cycle which may consume more energy but assure performance level QoS on all applications is used.

We point out that the *SVM* operates as a background process on Android and operates only with an explicit request from the application process in iOS. While managing both on-board and external sensors, the *SVM* does not operate until an application sends a request so that it can minimize the additional power draw that a hardware controlling module introduces. The size of the SVM software on a smartphone is ~1.20 MB on Android and ~2.30 MB on iOS and requires ~50 kB of memory while running with an additional ~20 kB of extra memory per each external sensor connected to the smart device. The SVM operates on Android version 4.3 (API Level 18) and iOS 5 or higher to support BLE connections.

As a way to open these local IoT resources to even more applications beyond a device's internal applications, the APIs provided by *SVM* and the data resources from the *SVM* application framework are shared using a Google App Engine-based service. As an identifier for each mobile computing platform, we use a tuple ID of `[GPS-location,mobile-platform-ID]`, where the associated phone number of the mobile platform or the MAC address is used as the `mobile-platform-ID`. This tuple ID is associated with the APIs that the mobile platform provides (Figure 2(b)) and allows other mobile platforms to access a remote mobile platform's local IoT resources when properly authenticated.

Finally, Figure 3 shows the overall architecture and usage scenario that we envision; the *SVM* on a mobile device manages its local IoT devices and exposes them to the cloud via the *SVM* application server. These IoT devices as well as virtual IoT devices (which we explain below) can be accessed not only by multiple applications running on that mobile device but also by remote applications running on various Internet-connected platforms.

2.2. Creating Virtual IoT Device: Sensor Data Mash-Up. The capability to easily access data from various external IoT devices, both locally and remotely, opens the possibilities to generate new information from the original data. For example, the capability to access temperature and humidity levels from two different sensors allows the generation of a new data type called "comfort level." Although it is possible to implement a dedicated application on a smartphone to read individual local sensors and calculate this comfort level, *SVM* takes a different approach where a virtual IoT device with "comfort level" sensor is created. The advantage of this approach is that any application running on a mobile device or on the Internet can access this shared information without dedicated connections to the physical sensors or individual knowledge of how they are retrieved or how the new information is calculated. Our *SVM*'s open APIs allow the generation of such new types of information sources by providing an `addDevice()` function. Using this functionality, as Figure 4 shows, data from different IoT devices can be "mashed up" to generate new custom data using a user

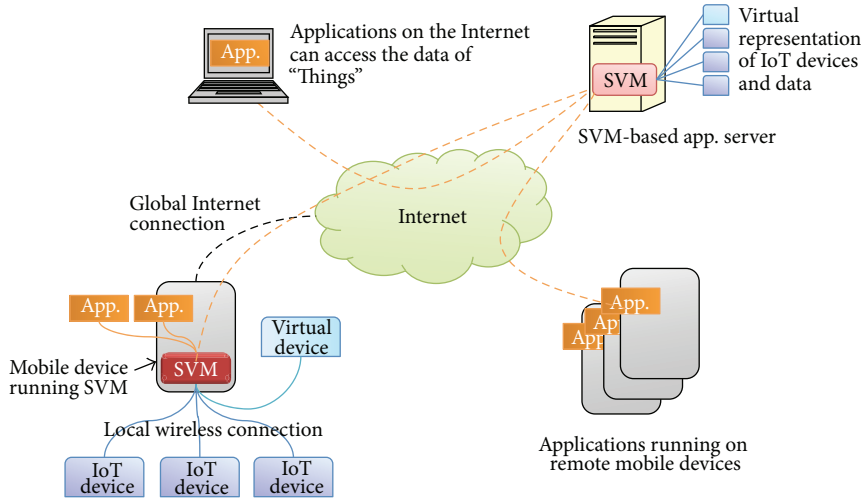


Figure 3: *SVM* usage scenario.

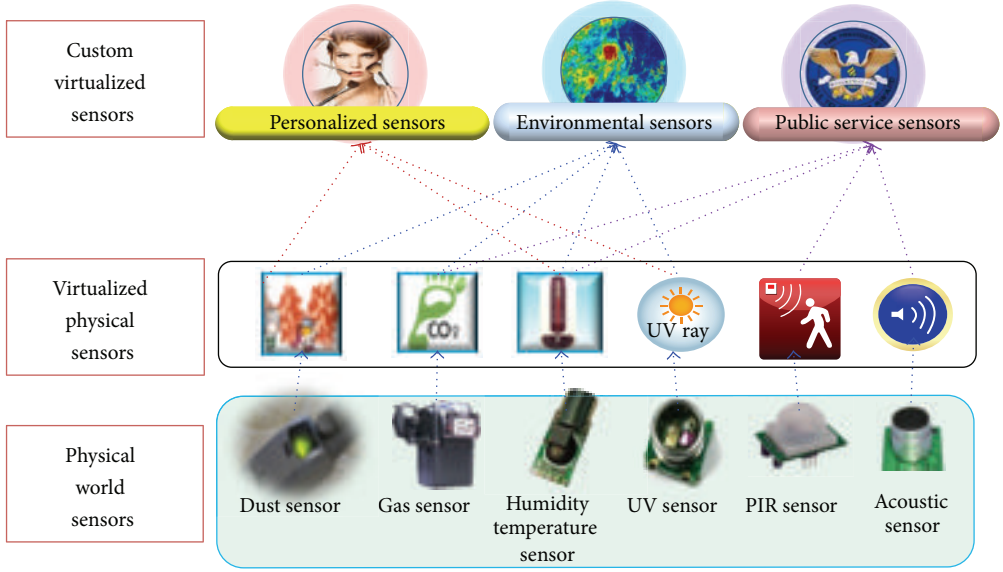


Figure 4: Custom virtualized device generation process using data from physical world sensors.

defined fusion function. Furthermore, as we illustrate in Figure 5, this new data resource can also be exposed as a separate customized virtual IoT device using the open APIs for various remote applications to access. For example, if a mobile or web application wishes to access environmental information from a specific location, it can access a virtual environment sensor instead of trying to connect to several physical sensors individually and reimplement the fusion function.

2.3. Software Reprogramming. The management of IoT devices from mobile platforms offers opportunity for any software updates required by the IoT devices to be achieved through the mobile platform. Furthermore, when a mobile device first enters a field of IoT devices, a new installation of software may be required on the mobile device to fully manage or utilize the features of the IoT devices required by the target application. For this purpose, *SVM* provides support for updating software on a target IoT device and also for installing required software on the mobile platform to utilize the IoT devices that are connected to themselves.

(i) IoT Device Software Update. *SVM* supports reprogramming of the external IoT devices as follows. First step, which occurs during the application development and packaging phase, is to combine the executable binaries generated for the IoT devices into a single smartphone application installation package. Fortunately, both Android and iOS development environments provide directories where raw files can be packaged as a single application installation file. Specifically, these designated folders are the `res/raw/` folder within Android IDE and the `documents/` directory for iOS. Once the IoT devices' binaries and the smartphone installation

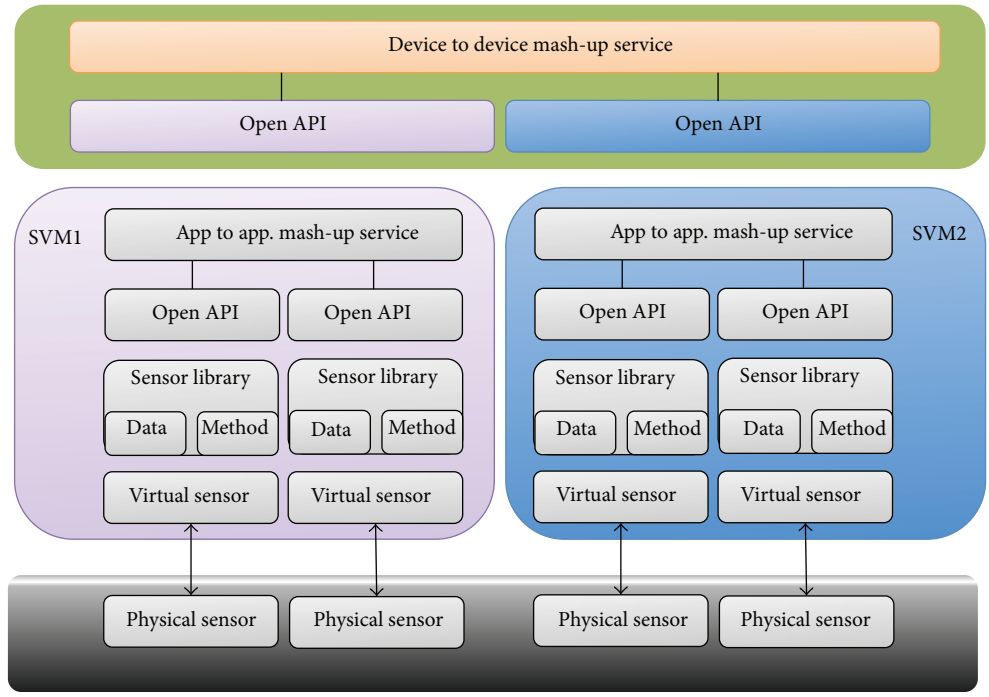


Figure 5: Illustration of sensor data mash-up procedure using *SVM*'s open APIs.

binaries are packaged and sent to the *SVM* (via URL), *SVM* extracts the binaries for the IoT devices and distributes them to the individual devices. At this step, we expect the IoT devices to incorporate some form of over-the-air bootloading and reprogramming method to allow the proper update of software. A well-known example of this is the iOS update process on iPhones. Furthermore for security reasons the mobile device that is reprogramming the IoT devices is authenticated and authorized by an application server before the binaries are distributed and installed.

(ii) Automatic Software Installation on Mobile Computing Platforms. Software update can also move in opposite direction where IoT devices push the required software to the mobile device that it is connected to. When a *SVM*-compliant IoT device with such reprogramming capability discovers and connects to a new mobile device, it simply pushes a predefined URL to the mobile device. Using this URL, the mobile platform retrieves the proper software that the IoT device requested and installs it while making the required configurations to interoperate with the *SVM*. Unless this software is installed on the mobile device, mobile device has limited access to the information and features provided by the IoT device.

3. Application Case Studies

Using the *SVM* environment, we now present two application case studies, implementation of two sample applications that benefit from the use of the *SVM* architecture.

3.1. Sensor Data Management and Mash-Up. The first application we designed with *SVM* is a *sensor data management* application. The goal of this application was to validate the effectiveness of using *SVM* for IoT device interaction and experimentally evaluate the functionality of IoT device resource management and the data mash-up functionality. Using the *SVM*'s open APIs as the development core, we were able to easily interconnect multiple external IoT platforms from the Android OS. As Figure 6 shows, our application interacts with nine different sensing modalities on three physically different IoT devices. We were able to access these devices using both Bluetooth and ZigBee communication modules attached to the smartphone's USB connector.

Furthermore, for testing the data mash-up functionality, we computed the comfort level of the current environment using the humidity and temperature data collected locally and an air clarity measurement of the area, which was accessible through GPS measurements and a web-based query to the meteorological services' server. Using a user defined algorithm to combine these measurements, we were able to create a new customized virtual IoT resource (comfort level) within the *SVM*. Based on the updates of each value, the value of this customized virtual IoT sensor was updated automatically on a periodic basis. Furthermore, this sensor was accessible not only from the smartphone that the physical IoT devices were connected to but also from a cloud-based application server from which other applications can retrieve data.

3.2. Home Appliance Management. Managing and controlling smart home appliances (e.g., air conditioner, smart

(a) Sensor value display (b) Device configuration (c) Mash-up example ("comfort level")

FIGURE 6: Screenshot of sensor data management application.

lighting, and thermostat) are an attractive IoT application. While some appliances may include WiFi radios to achieve global connectivity through preinstalled WiFi APs, such an infrastructure may not be available in all households. However, we noticed that some products had Bluetooth connectivity which allows pairing with a mobile device for appliance management. But using them as they are suggests that a mobile device (or any controller device) needs to be within Bluetooth communication range of the appliance to control it. With *SVM*, smart appliances in a home can be exposed to the cloud as IoT resources and the *SVM* open APIs would allow users to control these devices remotely, either on the web or also from another mobile device.

To exemplify this idea, with the help from our industrial collaborators, we have used a robot cleaner and an air purifier with Bluetooth connectivity and installed our *SVM* on an Internet-connected Bluetooth-enabled smart TV which acted as a gateway to support global connectivity for various smart appliances. The reason for using a smart TV was to showcase a scenario where a device residing at home provides connectivity, and your mobile phone is used to control those devices remotely when you are not at home. The APIs to control the appliances were exposed through our SVM *cloud service* that we implemented, which shows a list of accessible controls that are authorized for each *SVM*-compliant device. Figure 7 shows the screenshot of our cloud service designed to access the *SVM*'s global open APIs through the Internet. Using the cloud-based open APIs, we were able to design a home appliance controlling application that could start and stop home cleaning and air purifying remotely before reaching home to manually control the devices.

4. Related Work

As sensor-equipped smartphones become more prevalent, many new and interesting applications have emerged that make use of the sensors on a smartphone. For example, PEIR [17] is a personal environmental impact report platform for participatory sensing systems research, and Nericell [18] is a monitoring system for road and traffic conditions using mobile smartphones. Eriksson et al. proposed a mobile sensor network for road surface monitoring [19], and SoundSense [20] is a scalable sound sensing system for people-centric applications on mobile phones. Furthermore, Krieger et al. use smartphone sensors for urban tomography in social science research [21]. Abstractions and functionalities provided by our proposed SVM can be used to ease and expedite the development of such applications.

There are several prior works that aim to connect wireless sensors to the Internet by utilizing various Internet standards centered at IP/IPv6. For example, IEEE 802.15.4-based protocols are designed specifically to support interoperability with other already existing IP-based devices (e.g., IETF 6LoWPAN [22] and RPL [23–25]) and communicate with the larger Internet architecture to transport their data to consumers and services in the cloud [13]. Dunkels et al. also evaluated the performance of low-power IPv6 for IoT using the Contiki OS [11].

Another line of related work is programming framework (and APIs) for mobile devices that allow backend users to easily program and task mobile devices on the Internet for collecting sensor data. Medusa [26] proposes a programming system for crown-sensing, which provides a programming language with high-level abstraction for crowd-sensing tasks, and supports specifying various forms of human mediation in

Figure 7: Application diagram and screen capture of our smart appliance management application.

the sensing workflow. It partitions the tasks between the cloud and smartphones and also supports incentives and user specified controls on smartphone resource usage. Another work by Ravindranath et al. [27] also explores tasking smartphones and provides complex data processing primitives and profile-based compile time partitioning. Furthermore, AnonySense [28] is a privacy-aware tasking system for sensor data collection and in-network processing, PRISM [29] proposes a procedural programming language for collecting sensor data from a large number of mobile phones, and [7] is a tasking abstraction for tiered sensor network. However, their focus is on the programming language abstractions on mobile smartphones and does not support virtualization of external sensors or sensor data mash-up or sensor reprogramming capability.

5. Summary and Future Research Directions

The Sensor Virtualization Module proposed in this work provides applications with a common virtualized

environment where external IoT devices can be easily accessed from and via mobile computing platforms. We achieve this by abstracting the networking aspect of IoT devices through user-held mobile devices as a gateway and providing a set of open APIs and device reprogramming functionality, which simplifies the access to various IoT resources. We believe that this work is one of the first attempts to step away from the traditional "stovepipe" software model where only a dedicated software, service, or a limited set of APIs provided by the IoT device vendors are available to third-party application developers. By allowing developers to easily access resources from various IoT devices, we envision that a diverse set of applications can be developed and many users will easily experience the effectiveness of IoT systems in shorter time.

An important next step that we foresee is well defining a protocol for data exchange between the smartphone and external sensors. While various standards can allow the devices to communicate, depending on the physical sensors' initial configurations (e.g., offering push or pull based services or a predefined wireless channel configuration), the quality and stability of data gathering can vary. We argue that an application level standard should address this issue (e.g., defining the format of physical sensor profiles). Furthermore, by providing modular environment for designing new virtual sensors, we believe that applications can maximize the usage of physical wireless sensors for designing various personalized services.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported by Project no. 10041725, "Development of Application Service and Software to Support Sensor Terminals for Providing Personalized Service Based on Smart Devices," from the Korean Ministry of Knowledge Economy. For Jeongyeup Paek, this research was supported by the Chung-Ang University Research Grants in 2015.

References

[1] E. Ancillotti, R. Bruno, and M. Conti, "The role of the RPL routing protocol for smart grid communications," *IEEE Communications Magazine*, vol. 51, no. 1, pp. 75–83, 2013.

[2] Cisco, "Smart Grid—Field Area Network," http://www.cisco.com/web/strategy/energy/field_area_network.html.

[3] V. C. Gungor, D. Sahin, T. Kocak et al., "A Survey on smart grid potential applications and communication requirements," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 28–42, 2013.

[4] T. Heo, K. Kim, H. Kim et al., "Escaping from ancient Rome! applications and challenges for designing smart cities," *Transactions on Emerging Telecommunications Technologies*, vol. 25, no. 1, pp. 109–119, 2014.

[5] A. Brandt, J. Buron, and G. Porcu, "Home automation routing requirements in low-power and lossy networks," RFC 5826, IETF, 2010.

[6] J. Martocci, P. De Mil, N. Riou, and W. Vermeylen, "Building automation routing requirements in low-power and lossy networks," Tech. Rep. RFC 5867, 2010.

[7] J. Paek, B. Greenstein, O. Gnawali et al., "The tenet architecture for tiered sensor networks," *ACM Transactions on Sensor Networks*, vol. 6, no. 4, article 34, 2010.

[8] J. Paek, J. Hicks, S. Coe, and R. Govindan, "Image-based environmental monitoring sensor application using an embedded wireless sensor network," *Sensors (Switzerland)*, vol. 14, no. 9, pp. 15981–16002, 2014.

[9] German Federal Ministry of Education and Research, Project of the Future: Industry 4.0, http://www.bmbf.de/en/19955.php.

[10] V. C. Gungor and G. P. Hancke, "Industrial wireless sensor networks: challenges, design principles, and technical approaches," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 10, pp. 4258–4265, 2009.

[11] A. Dunkels, J. Eriksson, N. Finne et al., "Low-power IPv6 for the internet of things," in *Proceedings of the 9th International Conference on Networked Sensing Systems (INSS '12)*, pp. 1–6, June 2012.

[12] J. W. Hui and D. E. Culler, "Extending IP to low-power, wireless personal area networks," *IEEE Internet Computing*, vol. 12, no. 4, pp. 37–45, 2008.

[13] J. G. Ko, S. Dawson-Haggerty, D. E. Culler, J. W. Hui, P. Levis, and A. Terzis, "Connecting low-power and lossy networks to the internet," *IEEE Communications Magazine*, vol. 49, no. 4, pp. 96–101, 2011.

[14] H. Kelly, "Helping 'smart' devices talk to each other," *CNN*, 2014.

[15] Y. Park, J. Yu, J.-G. Ko, and H. Kim, "Software radio on smartphones: feasible?" in *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications (HotMobile '14)*, ACM, February 2014.

[16] J. H. Lim, A. Zhan, J. Ko, A. Terzis, S. Szanton, and L. Gitlin, "A closed-loop approach for improving the wellness of low-income elders at home using game consoles," *IEEE Communications Magazine*, vol. 50, no. 1, pp. 44–51, 2012.

[17] M. Mun, S. Reddy, K. Shilton et al., "PEIR, the personal environmental impact report, as a platform for participatory sensing systems research," in *Proceedings of the 7th ACM International Conference on Mobile Systems, Applications, and Services (MobiSys '09)*, pp. 55–68, June 2009.

[18] P. Mohan, V. N. Padmanabhan, and R. Ramjee, "Nericell—using mobile smartphones for rich monitoring of road and traffic conditions," in *Proceedings of the 6th ACM Conference on Embedded Networked Sensor Systems (SenSys '08)*, pp. 357–358, November 2008.

[19] J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madden, and H. Balakrishnan, "The Pothole Patrol: Using a mobile sensor network for road surface monitoring," in *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, pp. 29–39, ACM, June 2008.

[20] H. Lu, W. Pan, N. D. Lane, T. Choudhury, and A. T. Campbell, "Soundsense: scalable sound sensing for people-centric applications on mobile phones," in *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services (Mobisys '09)*, pp. 165–178, ACM, Wrocław, Poland, June 2009.

[21] M. H. Krieger, M.-R. Ra, J. Paek, R. Govindan, and J. Evans-Cowley, "Urban tomography," *Journal of Urban Technology*, vol. 17, no. 2, pp. 21–36, 2010.

[22] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 packets over IEEE 802.15.4 networks," RFC 4944, 2007.

[23] T. Winter, P. Thubert, A. Brandt et al., "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," RFC Editor RFC6550, 2012.

[24] J. Ko, J. Jeong, J. Park, J. A. Jun, O. Gnawali, and J. Paek, "DualMOP-RPL: supporting multiple modes of downward routing in a single RPL network," *ACM Transactions on Sensor Networks*, vol. 11, no. 2, article 39, 18 pages, 2015.

[25] H.-S. Kim, J. Paek, and S. Bahk, "QU-RPL: queue utilization based rpl for load balancing in large scale industrial applications," in *Proceedings of the 12th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON '15)*, Madison, Wis, USA, June 2015.

[26] M.-R. Ra, B. Liu, T. F. La Porta, and R. Govindan, "Medusa: A programming framework for crowd-sensing applications," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys '12)*, pp. 337–350, ACM, June 2012.

[27] L. Ravindranath, A. Thiagarajan, H. Balakrishnan, and S. Madden, "Code in the air: simplifying sensing and coordination tasks on smartphones," in *Proceedings of the 13th Workshop on Mobile Computing Systems and Applications (HotMobile '12)*, February 2012.

[28] C. Cornelius, A. Kapadia, D. Kotz, D. Peebles, M. Shin, and N. Triandopoulos, "AnonySense: privacy-aware people-centric sensing," in *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services (MobiSys '08)*, pp. 211–224, June 2008.

[29] T. Das, P. Mohan, V. N. Padmanabhan, R. Ramjee, and A. Sharma, "PRISM: platform for remote sensing using smartphones," in *Proceedings of the 8th Annual International Conference on Mobile Systems, Applications and Services (MobiSys '10)*, pp. 63–76, June 2010.