2013 IEEE 10th International Conference on Ubiquitous Intelligence & Computing and 2013 IEEE 10th International Conference on Autonomic & Trusted Computing

Supporting Localized Interactions among Heterogeneous Smart Things with ThingWare

Junjun Kong, Yao Guo, Xiangqun Chen and Weizhong Shao
Key Laboratory of High-Confidence Software Technologies (Ministry of Education)
School of Electronics Engineering and Computer Science, Peking University, Beijing, China
{*kongjj07, yaoguo, cherry, wzshao*}*@sei.pku.edu.cn*

***Abstract*—The development of ubiquitous computing has witnessed the invention of various smart things (or smart objects), which normally refer to conventional devices equipped with sensing, computing, and communication capabilities. Programming with smart things faces many challenges because they are mobile, dynamic, and heterogeneous. This paper proposes ThingAPI as a uniform programming abstraction to support efficient programming with heterogeneous smart things. ThingAPI tries to accomplish two objectives: *making local smart things interactive, and making heterogeneous smart things easily programmable*. ThingAPI allows applications on smart things to interact with a wide range of things and flexibly extend their functionalities. We have built a lightweight middleware prototype, ThingWare, to implement ThingAPI over several platforms. With ThingWare, we can easily program household and office appliances such as TV sets, air-conditioners, projectors, *etc*. To demonstrate the feasibility and efficiency of our approach, we construct a representative application and present detailed analysis of various performance results.**

***Keywords*-Internet of Things, smart things, programming abstraction, localized interaction**

I. Introduction

Since the introduction of ubiquitous computing by Mark Weiser more than twenty years ago, many pervasive systems and smart devices with "invisible" computing abilities have been built. Typical examples of these objects/products range from large electronic appliances such as smart TV and smart refrigerator, to tiny electronic equipments such as smart light bulbs [1], [2], and even traditional attires including smart shoes and smart clothes. In recent years, many people have been referring these objects as *smart objects* or *smart things* [3], [4], [5].

These existing or emerging smart things are typically enhanced by computation and communication technologies. Therefore, they are able to offer many new extensible features compared with their conventional built-in functionalities. With the development of Internet of Things (IoT) technologies [6], many promising applications can be realized with smart things, such as home appliance control [7], smart transportation and logistics [4], *etc.*

Developing complex pervasive applications with smart things requires a variety of expertise, ranging from device interfaces and networking protocols to programming languages and operating systems. These issues have been studied extensively in various pervasive computing environments. As a result, a number of middleware systems have been proposed to solve the problems [8], [9]. Some recent examples of these systems include TeenyLime [10], Prottoy and FedNet [11], HomeOS [7], *etc.* These middleware systems typically deal with either a fixed set of smart devices, or applications in a specific domain. They are largely centralized computing information/control systems with the ability to manage a set of digital electronic devices (such as HomeOS and FedNet), sensors and actuators (mostly WSN middleware), *etc*.

However, most of these pervasive systems have been focused on dealing with traditional devices instead of smart things. Smart things invented in recent years exhibit a very different outlook compared to traditional pervasive systems. As more and more daily things being transformed into smart things [12], [3], we face new challenges to develop pervasive applications with smart things of powerful communication and computation capabilities.

Because most smart things are equipped with some sort of communication modules, it becomes necessary to provide localized interactions among them, such that spatially nearby smart things have more opportunities to directly interact with each other. Realizing these functionalities requires the devices possess the ability to interact with each other. For example, a mobile smartphone could be equipped the ability to control nearby smart appliances such as smart light, smart TV, smart air conditioner, *etc.*

On the other hand, the heterogeneity of various smart things makes it difficult to manage and program them. Different types of smart things are highly heterogeneous in terms of their built-in functionalities, computational capabilities, underlying networks and protocols, operating systems, supported programming languages, *etc*. Unlike traditional PCs, smart things are diverse. A typical example could involve that an Android smartphone interacts with a nearby smart lamp equipped with totally different hardware and software [1], [2].

We propose ThingAPI as a uniform solution to program heterogeneous smart things. ThingAPI aims to achieve two objectives: *making local smart things interactive, and*

978-1-4799-2481-3/13 $31.00 © 2013 IEEE
DOI 10.1109/UIC-ATC.2013.102

making heterogeneous smart things easily programmable. Our approach consists of three major components:

- *Localized interaction*: We extend smart things to support a localized interaction mechanism, which enables mart things to identify each other and access functionalities of each other, without necessarily being coordinated by a central unit.
- *ThingAPI*: We propose ThingAPI as a unified *programming abstraction* to address the heterogeneity issue of smart things. With this abstraction, developers can easily program multiple smart things without knowing the low-level details of smart things.
- *ThingWare*: ThingWare provides runtime support for ThingAPI. It is deployed on each smart thing as the underlying software infrastructure. Applications on these smart things are running above the ThingWare.

In order to demonstrate the feasibility and capability of ThingAPI, we have built a lightweight middleware prototype (*i.e.*, ThingWare[1]) to implement ThingAPI programming abstraction over a variety of smart things, including IRIS-mote-powered devices, SunSpot-powered appliances, Android smartphones, and laptop computers. The current implementation of ThingWare is able to support the programming of household and office appliances. We also validate our approach with a proof-of-concept case study and quantitative evaluation of the implemented prototype.

II. A Localized Interaction Model

We introduce a localized interaction model to enable spatially neighboring smart things to interact with each other. One smart thing can dynamically discover a set of smart things within its proximity. It can then query services provided by any of them on the fly. If necessary, one smart thing can directly utilize a list of functions of another nearby smart thing by simply invoking certain services exported by that smart thing. In this way, applications can be built over a dynamic set of underlying smart things.

LIN: We assume that smart things possess the ability to directly interact with each other and even human beings under some circumstances. Through decentralized and localized interaction, each smart thing can dynamically build a ***l**ocal **i**nteraction **n**etwork* (LIN), which normally covers one-hop neighboring nodes (*i.e.*, surrounding smart things). The local interaction network integrates several different wireless networks including Bluetooth, ZigBee, and Wi-Fi. Any surrounding smart things within one-hop communication range of Bluetooth, ZigBee, or Wi-Fi will fall into the coverage of one LIN. Taking a smartphone with Wi-Fi and Bluetooth enabled as an instance, it can dynamically build its (logical) LIN by detecting Bluetooth neighbors and one-hop reachable nodes via Wi-Fi. The LIN provides a communication base for nearby smart things to recognize and connect with each other.

[1]Coincidentally, there is an open-source firmware collection with the same name as "ThingWare". Our middleware prototype and the firmware are totally irrelevant.

Table I
ThingAPI interfaces

Interface	Semantic meaning
QueryThings	Query all active smart things in the vicinity;
ThingAttr	Retrieve all attributes of the smart thing specified by a smart thing ID;
ThingSvc	Retrieve all services of the smart thing specified by a smart thing ID;
ThingCall	Call a smart thing service by specifying smart thing's ID and service ID, along with proper operation arguments;
RegThingSvc	Register a local smart thing service to the underlying service manager;
UnregThingSvc	Unregister a specified local smart thing service from the ThingWare system;

However, LIN does not guarantee that all smart things understand and utilize each other's functions. Therefore, we introduce *STCP* and *Smart Thing Service* over the LIN. The smart thing service enables them to utilize functions provided by other smart things, even though they are inherently heterogeneous in terms of functionalities, network interfaces, and computing hardware.

STCP: We introduce the **S**mart **T**hing **C**ommunication **P**ipe (STCP) as a communication model for heterogeneous smart things, which acts like a TCP stream and connects two smart things. One STCP connects two smart things over the LIN between them. It can automatically select available networks (such as Wi-Fi, Bluetooth, ZigBee, *etc.*) and establish communication connections between two smart things by reusing existing network protocols. The STCP model provides an omni-connection solution for smart things to handle the heterogeneity of underlying communication networks.

Smart Thing Service: One major issue of developing applications with multiple smart things is their heterogeneity in terms of computational capabilities, built-in functions, networking protocols, and so on. In particular, different types of smart things differ in their built-in functions. For example, a smart lamp may have these built-in functions: (a) changing light color, (b) adjusting light brightness, *etc.*, while a smart air conditioner may have some totally different functions: (a) cooling/heating, (b) adjusting wind speed, and so on. To address this issue, we adopt a service-oriented approach. The built-in functions are encapsulated into *smart thing services*, and the functions of a smart thing can be used by other smart things via calling smart thing services over STCP.

III. ThingAPI Programming Abstraction

We propose ThingAPI as a feasible programming abstraction for supporting localized interactions among heterogeneous smart things.

The ThingAPI programming abstraction includes three major components: smart thing, smart thing services, and

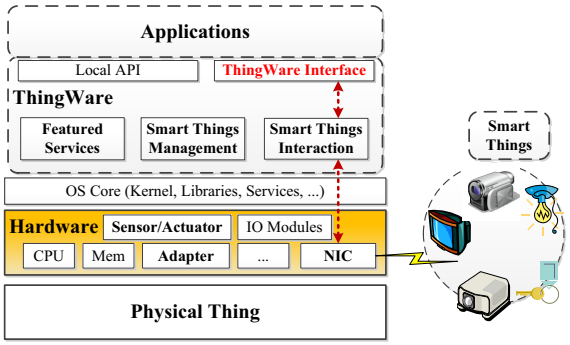


Figure 1. Five layers of a smart thing structure with ThingWare

ThingAPI programming interfaces. From the perspective of smart things programming, a smart thing is a first-order programming construct. The developer can directly use it as using objects in an object-oriented programming language. The ThingAPI programming interfaces define a set of methods manipulating smart things in the surroundings.

Table I lists the ThingAPI programming interfaces in detail. ThingAPI interfaces can be used to accomplish the following key tasks: (1) discovering smart things (*QueryThings*, *ThingAttr*); (2) accessing smart thing services (*ThingSvc*, *ThingCall*); (3) publishing / abolishing a smart thing service (*RegThingSvc*, *UnregThingSvc*). Using these interfaces, applications hosted by one smart thing can easily access the services exported by its surrounding smart things without knowing their details. Conventionally, a smartphone cannot directly interact with a table lamp, but when the lamp becomes a "smart lamp", it becomes possible for them to interact with each other. The smart lamp can export its functions as smart thing services such as `Turn On`, and the smartphone can easily control the smart lamp by simply accessing the `Turn On` service with ThingAPI interfaces.

IV. Design and Implementation

We design a lightweight middleware system called *ThingWare*, which provides support for both the smart things interaction mechanism and ThingAPI programming abstraction.

A. Design of ThingWare

Design Issues. Application developers may have difficulty in programming multiple heterogeneous smart things because of three aspects: (1) currently no ease-to-use programming interfaces available for developers, (2) the complexity of managing multiple dynamically changing smart things that have varied computing hardware and built-in functionalities, and (3) that the variety of underlying networks makes it difficult to connect two smart things efficiently and reliably.

ThingWare handles the management and hides the details of underlying hardware platforms and networks, so that application developers can focus on high-level application logic by leaving the low-level complexity to ThingWare. ThingWare handles three basic issues: (1) application-friendly programming interfaces for operating local device and nearby smart things, (2) management of both native built-in components and discoverable smart things in the vicinity, and (3) handling smart things interactions over multiple networks (covering LAN and PAN networks). In practice, ThingWare is implemented as a lightweight middleware system residing in each smart thing.

ThingWare Structure. As shown in Fig. 1, a smart thing with ThingWare inside is composed of five layers from bottom to top: physical thing (such as physical components of a thing and its built-in functionalities), computing hardware (*e.g.*, motherboard, networking cards, sensors, actuators, *etc.*), OS core (such as OS kernel, OS services, libraries, *etc.*), ThingWare, and applications. ThingWare includes these modules: local API, ThingWare interface, smart things interaction, featured services, and smart things management.

Local API provides platform-dependent programming interfaces associated with device-specific functionalities, and therefore, different smart things have different local APIs available. ThingWare interface provides platform-independent programming interfaces, which means different smart things have the same interfaces to interact with each other. Here the ThingWare interface module implements the ThingAPI programming abstraction by providing a fixed set of application programming interfaces (APIs).

The smart things interaction module realizes the STCP abstraction. This module enables applications to connect to surrounding things through multiple heterogeneous networks without knowing the details of the networks. The benefit of this abstraction is that applications can establish connections to other things easily and reliably, which can save application developers from wasting much time in handling error-prone network connections.

The featured services module manages the services provided by this local thing, and some of them are encapsulation of some built-in functionalities. Taking a smart TV as an example, some built-in operations such as *"turn on", "'volume up", "show next program"*, *etc.*, can be encapsulated into featured services of the smart TV.

The smart things management module manages discovered smart things, and maintains a list of presently discoverable smart things and attributes of each discovered smart thing. Thereby, when an application inquires the surrounding smart things via the *QueryThings* method in ThingAPI, this module can response immediately.

B. System Implementation

We have implemented a working prototype to demonstrate the feasibility of *ThingAPI*. The implemented lightweight middleware, **ThingWare**, realizes ThingAPI programming interfaces on mote-powered devices, Android-powered smartphones, and laptop computers. We used resource-limited IRIS motes and SunSpot nodes to transform daily things into smart things.

Localized interaction mechanism. Our smart things prototypes primarily use PAN (personal area network) and WLAN networks to achieve localized interaction. PAN networks do not rely on any networking infrastructure, thus they enable smart things to interact in an ad hoc manner. WLAN depends on AP (wireless access point) and some other infrastructure, however, it features high network bandwidth and low latency compared with PAN. Thus, WLAN provides a feasible way under some circumstances, such as sharing pictures and videos between an Android smart phone and a smart TV.

TinyOS-powered smart things (*e.g.*, IRIS smart sensors) use the *Active Message (AM)* to establish LIN networks in their vicinity. Meanwhile, SunSpot-powered smart things (*e.g.*, smart TV, smart air conditioner, and smart projector) use the *RadioGram* packets to create LIN networks around them. Both of them implement smart things discovery mechanism through broadcasting ZigBee packets. Android-supported smart things discover spatially-nearby things via UDP broadcast over Wi-Fi or SDP over Bluetooth. However, currently TinyOS-powered smart things cannot directly connect to SunSpot-powered ones, and therefore, we introduce a smart things gateway to interconnect them.

STCP implementation. STCP communication pipe is implemented by reusing several existing data transfer protocols. In the ThingWare middleware, the STCP management module handles all data exchange among smart things. It consists of three units: low-level data transfer, STCP buffer, and STCP stream. The data transfer unit is responsible for transferring STCP messages sequentially and reliably. This unit resides on top of several existing protocols such as TCP over Wi-Fi or LAN, RFCOMM over Bluetooth, RadioGram over ZigBee, *etc*. It detects the available network interfaces and delivers STCP messages with TCP, RFCOMM, and/or RadioGram. The STCP buffer unit temporarily stores all data payload to be sent or received from other smart things. The STCP stream provides a communication abstraction for applications, which has the same usage to TCP socket. It brings several benefits. First, this abstraction is familiar to many developers, and thus minimizes their effort of learning to use it. Second, it moves the complexity of handling several different protocols and networking interfaces to the underlying STCP message transfer unit, and thus hides the heterogeneity of networks from smart thing applications.

ThingAPI implementation. ThingAPI is implemented by providing methods or functions binding with different programming languages. TinyOS-powered smart things use nesC, while SunSpot-powered ones and Android smartphones use Java. ThingAPI provides a generic way to access smart thing services. Through dynamic service discovery and service invocation, smart things can flexibly manipulate each other. Compared with web services [13], smart thing service is a simple interaction mechanism optimized for resource-limited smart things. It supports both IP-based network and infrastructure-less ad-hoc network (such as personal area network), which makes it more feasible for things with heterogeneous network interfaces.

```
List<Thing> thingsList = null;
ThingsManager thingsManager = ThingsManager.getDefaultManager();
//get a list of surrounding things
thingsList = thingsManager.queryThings();
//filter out a smart thing of interest
int ac = ThingType.HOUSEWARE_AC; //this value is shared globally
Thing acThing = findThing(thingsList, ac);
if (acThing == null) { return; }
//get services provided by a smart thing
String thingID = acThing.getUuid();
List<ThingService> servicesList = thingsManager.thingSvc(thingID);
if (!findService(servicesList, "turnOnOff")) { return; }
//perform a service call on that thing
ThingServiceCall serviceCall = new ThingServiceCall("turnOnOff", null);
ThingServiceResult result = null;
try {
    result = thingsManager.thingCall(thingID, serviceCall);
} catch (ThingServiceException e) {
    e.printStackTrace();
}
```

Figure 2. Program code from an Android application using ThingAPI to remotely operate a nearby air conditioner (AC)

V. Evaluation

We first describe a proof-of-concept case study to demonstrate the feasibility of ThingAPI, and then present some numerical results to quantify the performance of ThingWare.

A. Case Study

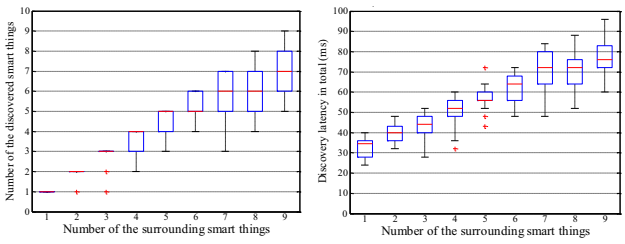
We describe how ThingAPI makes things easily programmable and interactive in this case study: manipulating household appliances with an application on a smartphone.

Development of the application. The application has two features: (1) dynamically detecting the surrounding smart things, and (2) automatically generating a control panel through discovering the services provided by the detected smart thing. Feature#1 can be implemented using *QueryThings* and *ThingAttr* interfaces, and Feature#2 can be simply done with *ThingSvc* and *ThingCall*. A user can select a service and invoke that service by clicking a button in the GUI window. This functionality can be achieved by writing about 10 lines of Java code basing on the ThingAPI programming interfaces. In Fig. 2, we list the core code snippet implementing the discovery of both smart things and smart thing services, and as well the invocation of the "turnOnOff" service provided by a smart AC (air-conditioner).

Dynamically finding newly added smart things. When new smart things (such as a smart TV) are added, our system can still work correctly. The newly added smart things will be automatically detected by the ThingWare infrastructure, and the end user does not need to manually configure them, which would ease the user greatly. Supposing a new smart TV is added, the application will list the newly added smart TV immediately, and generate a clickable itme to user for manipulating the TV.

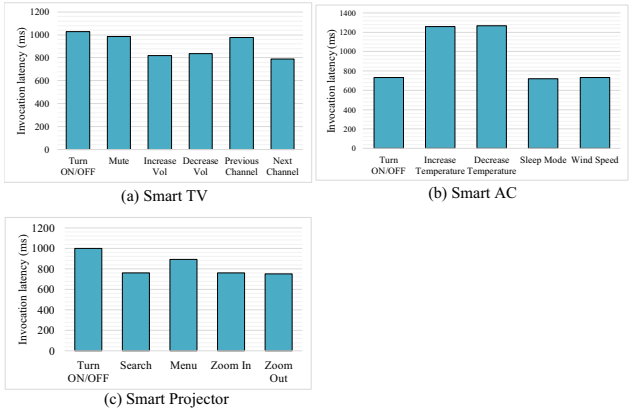
B. Performance Results

We conducted a set of experiments using IRIS sensor motes and SunSpot-powered smart devices on which our



(a) Total number of the discovered smart things (b) Total discovery latency for different cases

Figure 3. Distribution of the total discovered number and latency of multiple IRIS-mote-powered smart things.



(a) Smart TV (b) Smart AC

(c) Smart Projector

Figure 4. Average invocation latency of the smart TV, AC, and projector.

prototyped system *ThingWare* was deployed. We evaluate the performance of our prototype system from two aspects: (1) the coverage and latency of smart things discovery, and (2) the overhead of ThingAPI invocations.

Experimental setup. To collect performance data about the ThingWare prototype, we deployed the ThingWare prototype into IRIS (XM2110) sensor nodes and other smart things prototypes, and used a laptop computer to record performance logs. We built two benchmark applications to evaluate the performance of the prototype. One application (denoted as *AppBench1*) periodically issued query of things and service invocation to IRIS-powered smart things (smart sensors). The other application (denoted as *AppBench2*) periodically performed query of things and ThingAPI invocation to SunSpot-powered smart things (*i.e.*, smart TV, AC, and projector). Both of them ran on the laptop computer. When running *AppBench1*, each time we activated a certain number of smart sensors (from one to nine) which were placed within a lab room. While running *AppBench2*, we opened the smart TV, AC, and projector separately, and the three smart appliances were deployed in a meeting room.

Coverage and latency of discovery. Fig. 3(a) shows the distribution of total number of the discovered IRIS smart sensors, and Fig. 3(b) shows the distribution of the total discovery latency for different cases. Fig. 3 shows that (1) in some cases, all of the neighboring smart things placed in the room can be detected, and (2) from median values, in most cases the number of the discovered things is less than the real number of the surrounding things.

Table II
AVERAGE DISCOVERY AND INVOCATION LATENCY

Smart things	Discovery (ms)	Invocation (ms)
IRIS-powered (sense)	8	29.76
SunSpot-powered (actuate)	38.5	893.51

In Table II, we show the average latency of discovering one smart thing. The average discovery latency for the SunSpot-powered ones is around 38 ms (milliseconds), while it is approximately 8 ms for discovering one IRIS-powered smart thing. This difference is caused by adopting different mote operating systems and connection APIs.

Overhead of ThingAPI invocations. Table II also presents the average latency of ThingAPI invocation. This latency value means the time interval from calling one smart thing service to finally obtaining the execution result of that service execution. The average latency of one service invocation of the IRIS smart sensors is about 30 ms. This performance makes sense for sensing applications, since it can support more than 30 times of query per second.

In contrast, the average invocation latency on SunSpot-powered smart things is more than 890 ms. The value seems surprisingly large compared with the latency of mote-powered ones. It's because the IR Adapter module spends much time emitting infrared signal to control the corresponding appliance. Actually, we have tested the round-trip time of SunSpot RadioGram packets between *AppBench2* on a laptop and SunSpot node, and it's about 20 ms on average.

Fig. 4 presents the average ThingAPI invocation latency for smart TV, AC, and projector. Service invocation latency varies with respect to the types of smart thing services. Invoking the "`Turn ON/OFF`" service on smart TV takes more than 1000 ms on average, while ThingAPI invocation of some other services such as "`Increase Volume`" and "`Decrease Volume`" consumes less time, around 800 ms, as shown in Fig. 4(a). Meanwhile, Fig. 4(b) and Fig. 4(c) show similar observations.

In summary, the distinct variability of service invocation latency is a highlighted characteristic of ThingAPI. The inherent difference in the built-in functionalities of smart things leads to this variability.

VI. Related Work

In this paper, ThingAPI provides a flexible way to support direct localized interactions among smart things through appropriate programming abstraction. ThingAPI borrows many ideas and techniques from the prior work in spite of the difference in research focus.

Smart Objects/Things. Many kinds of smart devices have been proposed or invented both in the academia [14], [15] and industry [1], [2]. Many researchers use smart objects [16], [11], [17], [4] or smart things [3], [18] to

abstract and conceptualize different smart devices. Recently, Kortuem et al. define smart objects as "autonomous physical/digital objects augmented with sensing, processing, and network capabilities" [17].

Programmability of Things. Recently, several researchers have noticed the trend of daily objects' becoming smart objects and related programming issues [7], [19], [11]. Generally, they tried to answer the question how to build systems/applications over special-purpose computing devices. Kawsar et al. proposed a document-based declarative programming approach to building distributed smart objects systems [11]. Another closely related work is HomeOS by Microsoft Research [7], which proposed PC abstraction as their programming model for networked devices (akin to our smart things). However, they have some deficiencies, (1) that applications rely on centralized node which may become system bottleneck, and (2) that smart things could not fully interact with each other locally to complete some tasks within a small vicinity.

VII. Conclusion

The emergence and ubiquity of smart things encourage pervasive interactions among things. Those pervasive interactions can be achieved by appropriately programming smart things. However, programming smart things faces challenges because they are dynamic and inherently heterogeneous.

In this paper, we proposed ThingAPI, a programming abstraction for supporting localized interactions among heterogeneous smart things. We built a lightweight middleware prototype (ThingWare), to implement ThingAPI over a variety of smart things. We have shown that application developers can easily program household and office appliances with ThingWare. The evaluation results demonstrate the efficiency and usability of our approach.

Acknowledgment

This work is supported by the National Natural Science Foundation of China under Grant No.61103026; the National Basic Research Program of China (973) under Grant No.2011CB302604; the High-Tech R&D Program of China (863) under Grant No.2011AA01A202; the Science Fund for Creative Research Groups of China under Grant No.61121063.

References

[1] P. Bosua, "LIFX: The Light Bulb Reinvented," http://www.kickstarter.com/projects/limemouse/lifx-the-light-bulb-reinvented, 2012.

[2] Indiegogo, "Lumen Bluetooth LED Lightbulb," http://www.indiegogo.com/lumen, 2012.

[3] M. Kuniavsky, *Smart Things: Ubiquitous Computing User Experience Design*. Morgan Kaufmann, 2010.

[4] T. S. López, D. C. Ranasinghe, B. Patkai, and D. McFarlane, "Taxonomy, technology and applications of smart objects," *Information Systems Frontiers*, vol. 13, no. 2, pp. 281–300, 2009.

[5] T. Sánchez López, D. C. Ranasinghe, M. Harrison, and D. McFarlane, "Adding sense to the Internet of Things - An architecture framework for Smart Object systems," *Personal and Ubiquitous Computing*, vol. 16, no. 3, pp. 291–308, 2012.

[6] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[7] C. Dixon, R. Mahajan, S. Agarwal, A. J. Brush, B. Lee, S. Saroiu, and P. Bahl, "An operating system for the home," in *Proc. NSDI'12*. USENIX Association, 2012, p. 25.

[8] N. Mohamed and J. Al-Jaroodi, "A survey on service-oriented middleware for wireless sensor networks," *Service Oriented Computing and Applications*, vol. 5, no. 2, pp. 71–85, 2011.

[9] L. Mottola and G. P. Picco, "Programming Wireless Sensor Networks: Fundamental Concepts and State of the Art," *ACM Computing Surveys*, vol. 43, no. 3, p. 19, 2011.

[10] P. Costa, L. Mottola, A. L. Murphy, and G. P. Picco, "Programming Wireless Sensor Networks with the TeenyLime Middleware," in *Proc. Middleware 2007*, 2007, pp. 429–449.

[11] F. Kawsar, T. Nakajima, J. H. Park, and S.-S. Yeo, "Design and implementation of a framework for building distributed smart object systems," *The Journal of Supercomputing*, vol. 54, no. 1, pp. 4–28, 2010.

[12] S. Hodges, N. Villar, J. Scott, and A. Schmidt, "A New Era for Ubicomp Development," *IEEE Pervasive Computing*, vol. 11, no. 1, pp. 5–9, 2012.

[13] M. Bichler and K.-J. Lin, "Service-oriented computing," *Computer*, vol. 39, no. 3, pp. 99–101, 2006.

[14] M. Beigl, H.-W. Gellersen, and A. Schmidt, "Mediacups: experience with design and use of computer-augmented everyday artefacts," *Computer Networks*, vol. 35, no. 4, pp. 401–409, 2001.

[15] H. Gellersen, "Smart-Its: computers for artifacts in the physical world," *Communications of the ACM*, vol. 48, no. 3, p. 66, 2005.

[16] M. Beigl and H. Gellersen, "Smart-Its: An embedded platform for smart objects," in *Proc. Smart Objects Conference (SOC 2003)*, 2003, pp. 15–17.

[17] G. Kortuem, F. Kawsar, D. Fitton, and V. Sundramoorthy, "Smart objects as building blocks for the Internet of things," *IEEE Internet Computing*, vol. 14, no. 1, pp. 44–51, 2010.

[18] J. Ma, L. T. Yang, B. O. Apduhan, R. Huang, L. Barolli, and M. Takizawa, "Towards a smart world and ubiquitous intelligence: A walkthrough from smart things to smart hyperspaces and UbicKids," *International Journal of Pervasive Computing and Communications*, no. 1, pp. 53–68, 2005.

[19] F. Kawsar, K. Fujinami, and T. Nakajima, "Prottoy middleware platform for smart object systems," *International Journal of Smart Home*, vol. 2, no. 3, pp. 1–18, 2008.