# An Internet of Things (IoT) Architecture for Embedded Appliances

Takeshi Yashiro*, Shinsuke Kobayashi*, Noboru Koshizuka*†, and Ken Sakamura*†
* YRP Ubiquitous Networking Laboratory
28th Kowa Bldg., 2-20-1, Nishi-Gotanda
Shinagawa-ku, Tokyo 141–0031, Japan
† The University of Tokyo
7-3-1, Bunkyo-ku, Tokyo 113–8654, Japan
E-mail: {takeshi.yashiro@ubin.jp, shinsuke.kobayashi@ubin.jp, koshizuka@sakamura-lab.org, ken@sakamura-lab.org}

***Abstract*—Although much of the work has been done until today to realize the Internet of Things (IoT) into practice, most of the work focuses on resource-constrained nodes, rather than linking the existing embedded systems to the IoT network. In this paper, we propose the uID-CoAP architecture, a new architecture designed to host IoT services on common embedded systems, like usual consumer appliances. As they often need to provide a number of sophisticated functions compared to simple sensor nodes, we combine the constrained application protocol (CoAP) with the ubiquitous ID (uID) architecture. The latter plays a crucial role for keeping the knowledge and data required for practical complex IoT services. In addition, we provide a software framework for embedded appliance nodes, designed to reduce the burden of embedded appliance manufacturers by providing an intuitive, consistent, and easy-to-use API. Based on this idea, our framework provides functions to build RESTful services in addition to the low-level communication API. We have evaluated our system through a case-study, and showed that our framework can be used effectively to implement practical IoT applications over existing embedded systems with a small programming effort.**

## I. Introduction

The essential idea of the *Internet of Things (IoT)* has been around for nearly two decades, and has attracted many researchers and industries because of its great estimated impact in improving our daily lives and society. When *things* like household appliances are connected to a network, they can work together in cooperation to provide the ideal service as a whole, not as a collection of independently working devices. This is useful for many of the real-world applications and services, and one would for example apply it to build a smart residence; windows can be closed automatically when the air conditioner is turned on, or can be opened for oxygen when the gas oven is turned on. The idea of IoT is especially valuable for persons with disabilities, as IoT technologies can support human activities at larger scale like building or society, as the devices can mutually cooperate to act as a total system.

So far, much work has been done on realizing the IoT into practice [1]. Due to the efforts made earlier, the state-of-the-art IoT technology has matured to a certain extent, and several de jure and de facto standards have already been established. Under these circumstances, it is becoming more important than ever to build up a practical system design and implementation of the IoT technologies based on the achievements of these existing efforts.

Although the IoT technologies have evolved over recent years, most of the prior work aimed at adopting the IoT technologies for extremely resource-constrained nodes, like sensor network nodes that simply send collected data to base stations. On the other hand, little work has been done on applying IoT technologies into embedded devices around us including consumer appliances. However, as the purposes, complexities, and the underlying architectures are different between sensor nodes and consumer appliances, the existing frameworks designed solely for sensor nodes are not suitable for usual embedded devices. For example, the design of IoT middleware on event-driven operating systems like TinyOS [2] and Contiki [3] and real-time operating systems with multi-threading support like T-Kernel ([4], [5], [6]) shall apparently be different.

In this paper, we propose the uID-CoAP architecture, a new IoT framework that aims to provide a solution for this issue. That is, we propose a new way to let the existing embedded systems be integrated into the IoT network. For this purpose, we first present the IoT network architecture made up of two existing technologies: ubiquitous ID (uID) architecture and constrained application protocol (CoAP). The fundamental idea here is to build an IoT network made up of RESTful services, with the help of semantic knowledge back-end provided as the uID database. This semantic database is essential for the embedded appliance nodes to know how they can work together in cooperation. For simple sensor network nodes, simply sending data to or accepting requests from base stations would suffice, but for household embedded appliances, decision-making process on each node would become more complex. For this purpose, the uID database system provides an excellent solution for knowledge management required in IoT, by providing a unique identifier (called *ucode*) that is separate from network addresses.

In addition, we created a new software framework that helps manufacturers to append IoT functionalities on top of existing embedded systems. Unlike the prior work that targeted on simple sensor nodes, the proposed framework is primarily designed for consumer appliances. This makes the uID-CoAP software different from other IoT node frameworks from several aspects. Most notably, our framework is designed to provide an intuitive, consistent, and easy-to-use API for programmers of embedded systems so that the cost of adding IoT functions to their products is reduced. We have designed our framework on top of T-Kernel ([4], [5], [6]), a real-time operating system based on the design of Industrial TRON (ITRON) [7] used thoroughly as one of the de facto standards of embedded operating systems. Our framework is designed as a middleware on top of this operating system, which adds network communication functionalities required for the IoT. In our design, we decided to provide not only the simple low-level API, like send and receive, but also application-layer API to minimize the burden of application developers. Our API supports RESTful API [8] over CoAP (Constrained Application Protocol) [9], which allows engineers to add RESTful service on their products easily.

In order to evaluate the uID-CoAP architecture, we have created HEMS (home energy management system) application on this framework as a case study. Evaluation results showed that our framework can be effectively used to construct practical IoT applications over embedded appliances with a small programming effort.

The rest of this paper is organized as follows. In section II, the brief history of IoT in addition to our related work is shown. In section III, the overall design of the uID-CoAP architecture is discussed, while in section IV, the details of our software framework is explained. We evaluate the proposed architecture in section V through a case study, and finally this work is concluded in section VI.

## II. Related Work

Prior to our work, much work has been done on IoT for nearly two decades. The fundamental idea stems from the late 1980s as ubiquitous computing [10] and pervasive computing. The term *Internet of Things (IoT)* appeared in the late 1990s, and has been thoroughly studied from several aspects as discussed by Atzori et al in [1]. Their survey categorizes IoT paradigms into three: things-oriented, semantic-oriented, and the Internet-oriented visions. Our work here tries to enhance Internet-oriented approach with semantic-oriented method, both of which are required to build practical, complex IoT applications, which are expected on rich embedded devices.

So far, much work has been done to build a software framework for the IoT, as the concrete implementation for the IoT is needed for the actual deployment. Up to the present, much work has been done to realize these goals, many of which derive from sensor network projects that share the idea with the Internet-oriented IoT.

TinyOS [2] is an application-specific operating system platform for wireless sensor nodes. It is designed to be as compact as possible (fewer than 400 bytes) to be used on resource-constrained sensor network nodes like Mica [11] with 128KB of flash ROM and 4KB of RAM. Because of the fitness for such resource-constrained nodes, TinyOS has been used by many researchers as the basic platform for sensor network, and much middleware and many applications have been developed until now. They include but not limited to networking (e.g., [12]), database (e.g., [13]), and security (e.g., [14], [15]), all of which are useful in constructing IoT applications on sensor nodes. Similar discussion can be made on Contiki [3], which was developed as a sensor network operating system and now distributed as an open source operating system for the IoT. It is equipped with a number of good libraries and middleware for the IoT like [16], which are useful for IoT application development. Although Contiki provides multi-thread support called *protothread*, its functionalities are very limited and lack many of the essential features that real-time operating systems have.

For sensor network nodes designed to perform very simple tasks, like sending temperature data to the base station, these two provide excellent solutions for realizing the IoT nodes. On the other hand, both TinyOS [2] and Contiki [3] are not opted for building complex applications doing multiple tasks at the same time. As they are based on event-driven design, they cannot be used as replacements for real-time operating systems with threads in embedded systems. From here we conclude that a new framework is needed which can add IoT functions to such embedded systems.

As discussed previously in the Introduction, one of the key design policies of our work lies in the provision of RESTful API functions, not only the basic network-layer functions like send and receive. For the resource-constrained devices, much work has already been done as discussed in [17]. All the implementations introduced in this survey are not designed for embedded operating systems, and hence have the different design goals from ours, except for Erika OS CoAP [18]. However, the paper focuses on the idea of the CoAP-based REST transactions, and the details of is programming interface is not clearly described. Although similar application-layer API is provided for example in [16], this framework is insufficient to build complex IoT applications on embedded systems, because the request and the reply must be handled in a event-driven model. For more complex applications than a simple reading of non-waiting hardware sensor registers, mechanisms for supporting it are by all means needed.

## III. uID-CoAP Architecture

In this section, we briefly explain the uID-CoAP architecture, our overall IoT network architecture designed to support hosting of practical IoT services over embedded appliances.

### A. Overview

Basically, our IoT architecture can be considered as *ubiquitous ID (uID) architecture* [19] enhanced with concrete network mechanisms using CoAP (constrained application protocol) [9]. Fig. 1 shows the overall IoT network architecture. As can be seen in the figure, it comprises of not only the web of things, but also with cloud computing services that backs the IoT.

The basic idea is as follows. The IoT is made up of a number of PANs (personal area networks) which comprise parts of

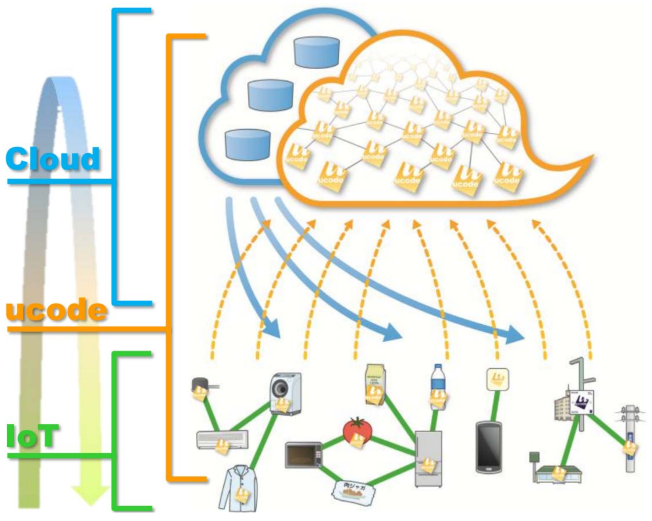

Fig. 1. Overall network architecture for the IoT

the IPv6 network. The information of the IoT nodes are kept on semantic database in the cloud. This database contains the node information including their profiles, relationship between others and so on. This information is crucial for the IoT nodes to know how they can work together in cooperation. For example, consider a situation where an air conditioner node wants to request all the window actuators to close the windows in the home. Of course, the requesting node can collect the network addresses of the nodes without any external database, but it cannot tell whether each device is a window, a fridge, or a light. In order for the IoT nodes to cooperate with each other, such knowledge data are considered essential for the IoT.

## B. uID Architecture

In our framework, we have adopted the uID architecture [19] for handling such knowledge because its design well suits our visions depicted in Fig. 1. The uID architecture can be considered as a backend for both the things-oriented and the semantic-oriented vision of the IoT discussed in [1], and is made up of two fundamental concepts: ucode and the uID database. These architectural components are accepted as recommendations in force by the ITU-T [20], and thus expected to be used broadly in the IoT system.

The *ucode* is a 128-bit identifier used to distinguish objects as well as notions involved the IoT. Unlike network addresses, ucodes are assured to be always unique, static, and never reused, all of which are necessary for the knowledge data using them to remain useful even after the network topology is changed, or even when the objects are removed from the network.

Using ucodes to specify resources in the IoT, the uID database keeps the semantic knowledge data required for the IoT. By using this database system, the IoT nodes can fetch the knowledge data required for their services in a unique, generalized, and persistent form. For this reason, we adopted this uID architecture as a basic model in our IoT architecture.

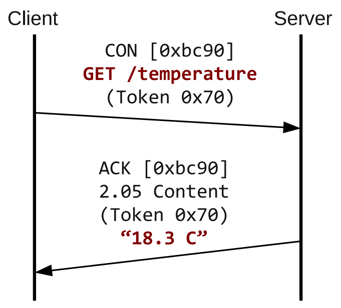

Fig. 2. An example of CoAP request and response

## C. Constrained Application Protocol (CoAP)

In order to give a concrete communication mechanism to the IoT framework, we accepted *CoAP (constrained application protocol)* [9] defined by the dedicated working group for it in the IETF. The main idea of this protocol is to provide a lightweight protocol for resource-oriented applications run on constrained networks. Its protocol design resembles that of hypertext transfer protocol (HTTP), on which the RESTful web services are implemented. Unlike HTTP, however, the communication is based on user datagram protocol (UDP) for reducing the communication cost, and its request/reply packet structure is made very compact to be used on low-power lossy networks like 6LoWPAN over IEEE 802.15.4 [21].

By using CoAP, access to IoT nodes can be done just like the using RESTful HTTP service on the web, however in a simpler, resource-efficient manner compared to HTTP. Fig. 2 shows an example of CoAP request and response. Unlike HTTP over TCP/IP, the UDP packet for `GET` method for resource "`/temperature`" is sent without any handshake, and the server replies the requested temperature data within a single packet. By combining these CoAP-based requests and replies, the IoT nodes can be made to work together through communications to provide an IoT service as a whole. Our framework therefore adopts CoAP as the communication protocol for constructing the IoT network.

## IV. uID-CoAP Software Framework

Based on the overall design of the IoT network architecture, now we would like to discuss the IoT software framework to be used on embedded systems.

Before going into the details, let us recall the basic principle of our framework: our framework is aimed for embedded appliances used around us, not for the resource-constrained nodes expected to spread through the IoT popularization. This principle made the design of the proposed framework different from others, which we discuss in later sections.

## A. Framework Structure

We have designed our framework on top of T-Kernel, a real-time operating system used commonly in embedded systems [5]. We have selected this platform because it supports a wide range of target hardware by providing multiple implementations that can be used for each specific range of

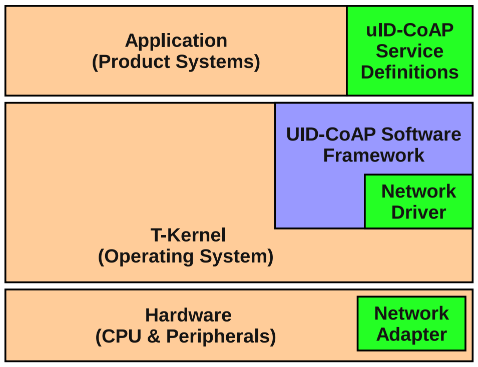

Fig. 3. Architecture of the uID-CoAP nodes

targets. For example, $\mu$T-Kernel [6] is designed to work within 8KB of ROM and 4KB of RAM when configured minimal, while for richer hardware, MP T-Kernel supports multi-core systems, T2EX (T-Kernel 2.0 Extension) supports filesystems and sockets, and TKSE (T-Kernel standard extension) supports processes and virtual memory.

Fig. 3 shows the architecture of uID-CoAP nodes. The primary concept of our design is to mitigate the burdens of manufacturers to add IoT communication functions to their existing products. This made our architecture to be created on existing embedded system architecture and providing our framework as an add-on to the existing system. The orange-colored parts in the figure comprise the original embedded system without any IoT connectivity, while purple- and green-colored parts are added anew to annex support for IoT communication. In order for developers to make RESTful service over the device, all they need to do is to create the components that are marked green in this figure. As the physical network interface and the network driver are in any case needed, so how easy the CoAP service can be defined is a big design challenge discussed next.

## B. uID-CoAP Service Definition

In order to simplify the CoAP service definition process, we decided to provide a network-independent, application-layer API for defining CoAP-based services. In order to fully abstract network communications from service definition, we took the approach that resembles virtual file system (VFS) used in modern operating systems.

Fig. 4 shows our CoAP interface definition structure in C language that product developers need to implement its instance to provide the desired service. As can be seen in the code, the structure is equipped with callback functions for CoAP methods, like `GET`, `PUT`, and `POST`. These callbacks does not require any network operations to be executed inside, and programmers can focus on data processing of requests (as inputs) to produce replies (as outputs).

In order to clarify this idea, we would like to give an example implementation of `GET` method callback function. Assuming that we want the air conditioner node to return the temperature data against the request for path "`/temperature`", the corresponding callback can be written as listed in Fig. 5. As can be confirmed in the code, the details of network protocols (including CoAP, UDP, and IP) are hidden to the programmers, and they are handled internally in the framework.

```
struct slpdev {
    /* Device identifier */
    ucode_t  devid;

    /* Extended information */
    void*    exinf;

    /* "GET" method implementation */
    INT  (*get)( struct slpdev* self,
                 const char* path,
                 void* buf, INT max );

    /* "PUT" method implementation */
    ER   (*put)( struct slpdev* self,
                 const char* path,
                 const void* buf, INT count );

    /* "POST" method implementation */
    INT  (*post)( struct slpdev* self,
                  const char* path,
                  const void* sndbuf, INT sndcnt,
                  void* rcvbuf, INT rcvmax );
    ....
};
```

Fig. 4. CoAP interface definition structure in C language

```
/* "GET" method implementation for
   specific air conditioner product */
INT aircon_get(self, path, buf, max)
   ....
{
   if (strcmp(path, "/temperature") == 0) {
      /* Read temperature from HW */
      UINT t = in_w(GPIO_TEMPERTURE) * C1 + C2;

      /* Output reply data to buffer */
      return snprintf(buf, max,
                "%d.%d C\n", t/10, t%10);
   }
   else if (strcmp(path, ....) == 0) {
      /* Can add other paths like this */
      ....
   }
   ....
   else {
      /* Non-existent path specified */
      return E_NOEXS;
   }
}
```

Fig. 5. Example implementation of GET method of an air conditioner

In order for applications to utilize CoAP interface definitions, several API functions are defined. Fig. 6 shows some important API functions provided for this purpose. In order to register the CoAP service and start accepting requests, programmers only need to call `slp_add_device` API function with appropriate service definition given as the argument. Then, our framework works in the background threads without affecting the existing embedded applications. This makes it easy for product manufacturers to add IoT functions to their

```
/* Start CoAP service */
ER    slp_add_device( struct slpdev* device );

/* Stop CoAP service */
ER    slp_remove_device( const ucode_t* target );

/* List up running CoAP services */
INT   slp_list_device( const ucode_t* start,
                       ucode_t* buf, INT max );

/* Get extended information for the device */
void* slp_get_exinf( const ucode_t* target );

....
```

Fig. 6. CoAP service control functions in the proposed framework

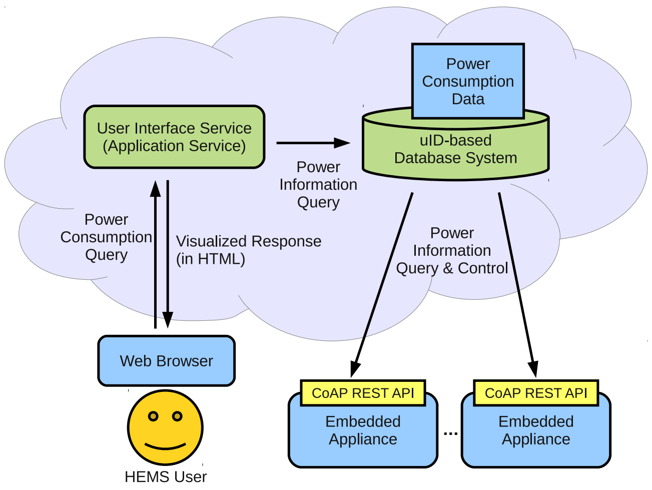

Fig. 7. HEMS architecture on the proposed framework

products, along with the fact that the service definitions can be done simply, because the existing product codes can be effectively reused.

Note here that in Fig. 4 and Fig. 6, ucode takes place as an identifier for the on-node service. This identifier is used for specifying the device uniquely even the network addresses are dynamic. This ucode information and the associated information are queried from uID database through a schema retrieval (in a CoRE link format) and stored as a part of device information in the database.

## V. Evaluation

In order to evaluate the uID-CoAP framework, we have implemented a sample embedded system equipped with IoT functions as a case study. We have selected HEMS (home energy management system) because the the home systems are composed of a number of consumer electronics, which well suits our visions of our architecture.

### A. System Overview

Fig. 7 shows the HEMS application structure on our proposed architecture. Based on the idea as described in section III, this application is made up on ucode and uID database system. User interface service is a HTTP application service

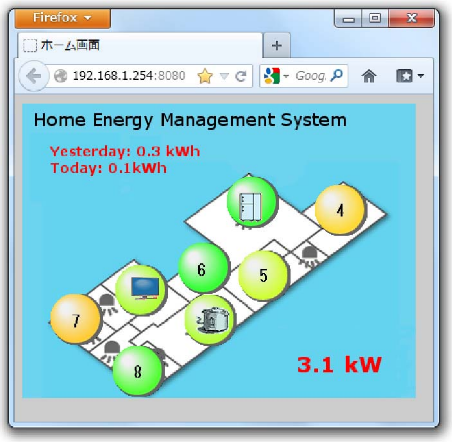

Fig. 8. Screenshot of our sample HEMS application

that queries the database for energy information, and returns the statistical data in a visualized form.

We have implemented this HEMS, of which screenshot is as given in Fig. 8. When there are much energy consumption in a device, this system notifies the user of the situation by turning the color to yellow and then to red. In addition to that, users can view power-time graph for each device, or can control the power status after transiting to the device-specific page by clicking the balloons displayed in the screenshot. Also note here that some of the balloons have icons in the figure. Such device-specific information used by applications is kept in the uID database system after retrieval. In this HEMS application, not only the icon images, but also the available CoAP resources, names of the devices, categories, and their electrical characteristics are kept in this database by linking them to ucodes.

Moreover, our system does not require additional meter modules to be installed in house, unlike many of the commercial HEMS like [22]. This is because our system estimates power consumption by software, based on the CPU idleness information and device driver statuses. This can be considered as one of the benefits accomplished by adopting our architecture model of the IoT.

### B. Implementation

The framework proposed in this paper has been applied to embedded appliances in Fig. 7. In order to implement the HEMS application, we have defined the following paths in the device CoAP service definition:

- `/voltage`, `/current`, `/power` (`GET` method)
  - Get the the voltage, current, and the power consumed in the device, respectively. (For example, strings like “`203.9 mW`” are returned.)
- `/supply` (`GET` and `PUT` method)
  - Get or set the power supply status (on, off, and etc.)

By accessing this interface, the database system collects the device power statuses by issuing `GET` method for `/power`, and stores the results as database entries. Then, when a application service has issued a query to this database, the uID database computes the aggregated data, which are to be converted to a visualized format by the user interface service. By accessing to this service, HEMS users can view the home energy statistics through a web browser as shown in Fig. 8.

In order to evaluate the size of our implementation, we have measured ROM and RAM usage of our framework (including the CoAP implementation) and the service definition of HEMS, respectively. We have implemented the HEMS system on a ARM1176 processor, and the resulting size of the framework consisted of 6KB of ROM and 4KB of RAM, while the HEMS service definition was implemented using 11KB of ROM and 9KB of RAM. These sizes are considered reasonable with respect to embedded real-time operating system kernels whose size ranges between 5KB and 200KB. Note here that the embedded Linux is not considered a replacement for our T-Kernel-based framework, as the embedded Linux is used for larger systems with several megabytes of memory. From the results presented here, we can conclude that our framework can realize compact implementations of CoAP RESTful services on embedded systems.

## VI. Conclusion

In this paper, we have proposed a new IoT architecture that lets existing embedded systems be integrated into the IoT network. This work differs from other work in that the framework is designed to be adapted to existing embedded systems, not only for sensor nodes with very simple functions and extremely limited resources. In order to realize this objective, we have combined the uID architecture and CoAP to host complex IoT applications requiring external knowledge. For reducing the burdens of manufacturers, we have designed our software framework for embedded system nodes to allow IoT service development with minimal efforts. As this framework supports application-layer API, which do not affect the existing codes and hides network-layer functions, product manufacturers only need to append a simple CoAP service definition, network driver, and physical network adapter to start IoT services on nodes.

In order to evaluate our system, we have implemented HEMS on top of this framework as a case study. Evaluation results showed that our architecture can realize practical IoT applications over existing embedded systems with minimal efforts.

## Acknowledgment

This work is partially supported under the promotion program for reducing global environment load through ICT innovation (PREDICT), the Ministry of Internal Affairs and Communications, Japan.

## References

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2010.05.010

[2] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer *et al.*, "TinyOS: An operating system for sensor networks," *Ambient intelligence*, vol. 35, 2005.

[3] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, 2004, pp. 455–462.

[4] J. Krikke, "T-Engine: Japan's ubiquitous computing architecture is ready for prime time," *Pervasive Computing, IEEE*, vol. 4, no. 2, pp. 4–9, 2005.

[5] T-Engine Forum, "T-Kernel 2.0," http://www.t-engine.org/.

[6] M. Kamio, K. Nakamura, S. Kobayashi, N. Koshizuka, and K. Sakamura, "Micro T-Kernel: A low power and small footprint RTOS for networked tiny devices," in *Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations*, ser. ITNG '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 587–594. [Online]. Available: http://dx.doi.org/10.1109/ITNG.2009.242

[7] H. Monden, "Introduction to ITRON the industry-oriented operating system," *IEEE Micro*, vol. 7, no. 2, pp. 45–52, Mar. 1987. [Online]. Available: http://dx.doi.org/10.1109/MM.1987.304844

[8] L. Richardson and S. Ruby, "RESTful web services," 2007.

[9] Z. Shelby, K. Hartke, and C. Bormann, "Constrained application protocol (CoAP)," 2013.

[10] M. Weiser, "The computer for the 21st century," *Scientific American*, vol. 265, no. 3, pp. 94–104, 1991.

[11] J. Hill and D. Culler, "Mica: a wireless platform for deeply embedded networks," *Micro, IEEE*, vol. 22, no. 6, pp. 12–24, 2002.

[12] T. Luckenbach, P. Gober, S. Arbanowski, A. Kotsopoulos, and K. Kim, "TinyREST - a protocol for integrating sensor networks into the internet," in *in Proc. of REALWSN*, 2005.

[13] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TinyDB: an acquisitional query processing system for sensor networks," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 122–173, Mar. 2005. [Online]. Available: http://doi.acm.org/10.1145/1061318.1061322

[14] C. Karlof, N. Sastry, and D. Wagner, "TinySec: a link layer security architecture for wireless sensor networks," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, ser. SenSys '04. New York, NY, USA: ACM, 2004, pp. 162–175. [Online]. Available: http://doi.acm.org/10.1145/1031495.1031515

[15] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar, "SPINS: security protocols for sensor networks," in *Proceedings of the 7th annual international conference on Mobile computing and networking*, ser. MobiCom '01. New York, NY, USA: ACM, 2001, pp. 189–199. [Online]. Available: http://doi.acm.org/10.1145/381677.381696

[16] M. Kovatsch, S. Duquennoy, and A. Dunkels, "A low-power CoAP for Contiki," in *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on*, 2011, pp. 855–860.

[17] B. C. Villaverde, D. Pesch, R. De Paz Alberola, S. Fedor, and M. Boubekeur, "Constrained application protocol for low power embedded networks: A survey," in *Proceedings of the 2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, ser. IMIS '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 702–707. [Online]. Available: http://dx.doi.org/10.1109/IMIS.2012.93

[18] E. Gutierrez Mlot, S. Bocchino, A. Azzara, M. Petracca, and P. Pagano, "Web services transactions in 6LoWPAN networks [1]," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2011 IEEE International Symposium on a*, 2011, pp. 1–2.

[19] N. Koshizuka and K. Sakamura, "Ubiquitous ID: Standards for ubiquitous computing and the Internet of Things," *Pervasive Computing, IEEE*, vol. 9, no. 4, pp. 98–101, 2010.

[20] ITU-T, *Multimedia information access triggered by tag-based identification*, International Telecommunication Union Recommendation H.642, Jun. 2012.

[21] G. Mulligan, "The 6LoWPAN architecture," in *Proceedings of the 4th workshop on Embedded networked sensors*, ser. EmNets '07. New York, NY, USA: ACM, 2007, pp. 78–82. [Online]. Available: http://doi.acm.org/10.1145/1278972.1278992

[22] Nippon Telegraph and Telephone East Corporation, "FLET'S miruene," http://flets.com/eco/miruene/.