2011 IEEE International Conference on Services Computing

Application Framework for Efficient Development of Sensor as a Service for Home Network System

Masahide Nakamura[1], Shuhei Matsuo[1], Shinsuke Matsumoto[1], Hiroyuki Sakamoto[1] and Hiroshi Igaki[2]
[1] *Graduate School of System Informatics, Kobe University, JAPAN*
[2] *Tokyo University of Technology, JAPAN*
Email: masa-n@cs.kobe-u.ac.jp

***Abstract*—The sensor as a service is an emerging application of the services computing. However, how to implement such sensor services efficiently and reliably is an open issue. This paper presents an application framework, called *Sensor Service Framework (SSF)*, that supports developers to build and deploy sensor services in the home network system (HNS). The SSF prescribes device-neutral features and APIs for the sensor devices to be deployed as Web services. Writing a small amount of code with the SSF, the developer can easily deploy any sensor device as a service in the HNS. The sensor service can provide a standardized access to heterogeneous sensor devices, as well as a context management service with user-defined conditions. We then present a *sensor mashup platform (SMuP)*, which allows the dynamic composition of the existing sensor services. To support non-expert developers, we also implemented a GUI front-end, called *Sensor Service Binder (SSB)*. The proposed technologies are implemented and evaluated in an actual HNS to demonstlate practical feasibility.**

***Keywords*-home network system, service-oriented architecture, context-aware services, application framework**

I. Introduction

Research and development of the *home network system* (HNS, for short) is recently a hot topic in the area of ubiquitous computing applications. Orchestrating house-hold appliances (e.g., TVs, DVDs, speakers, air-conditioners, lights, curtains, windows, etc.) via network, the HNS provides value-added services for home users.

Applying the *service-oriented architecture (SOA)* to the HNS is a smart solution to achieve the programmatic interoperability among heterogeneous and distributed appliances. Wrapping proprietary control protocols by Web services achieves loose-coupling and platform-independent access methods for external users and software agents. Several studies have been reported on the service orientation of home appliances (e.g., [1][2]). We have also been developing a service-oriented HNS, called *CS27-HNS*, using legacy home appliances [3]. The CS27-HNS is still evolving with new applications developped, such as smart remote controllers, voice controls, integrated services, an energy visualization system [4], a feature interaction manager [5].

Our next challenge is to deploy sensor devices in the HNS, in order to achieve sophisticated *context-aware services* [6]. Indeed, it is not difficult to develop proprietary applications, where sensors and appliances are tightly coupled. However, this approach lacks reusability and interoperability of sensors, which ruins the advantage of the service-oriented HNS.

A smarter approach is to apply the SOA to the sensors as well, namely, *sensor as a service* [7][8]. Many studies have been conducted related to the service orientation of the sensors and context-aware services (e.g., [9][10][11][12]). However, most existing methods emphasized features beneficial to service consumers only, such as service discovery, composition, and context reasoning. They often abstracted detailed implementation of the elementary sensor services. In reality, however, adapting a given sensor device to the SOA is not a trivial problem. Thus, how to implement good sensor services efficiently is not fully studied yet.

In this paper, we propose a practical application framework, called *Sensor Service Framework (SSF)*, to facilitate the implementation of the sensor as a service, especially for the service-oriented HNS. The SSF prescribes device-neutral APIs of the sensor devices to be deployed as Web services. From every sensor service, a client can obtain a normalized sensor value via `getValue()` method, regardless of the type of the sensor device. The SSF also implements a context management service. Using `register()` method, a client first registers a condition of a context to a sensor service. The sensor service periodically monitors the registered context, and notifies the client when the condition becomes true. Using `subscribe()` method, the client can bind the notification to any Web service in the HNS, which allows rapid creation of a user-defined context-aware service.

To extend the advantage of the SSF, we then propose the *sensor mashup platform (SMuP)*, enabling the *composition as a service* [13] within the SSF. Using the SMuP, one can dynamically create virtual composite sensor services from the existing sensor services. We also develop a GUI front-end of the SSF, called *Sensor Service Binder (SSB)*, to help non-expert users create own context-aware services.

We have implemented the proposed SSF by Java, and developed 16 kinds of sensor services using the SSF and Phidgets sensor devices [14]. These sensor services have been deployed in our CS27-HNS by Apache Axis2 Web services. The proposed methods are evaluated from practical feasiblity and validity of the approach.

978-0-7695-4462-5/11 $26.00 © 2011 IEEE
DOI 10.1109/SCC.2011.18

IEEE computer society

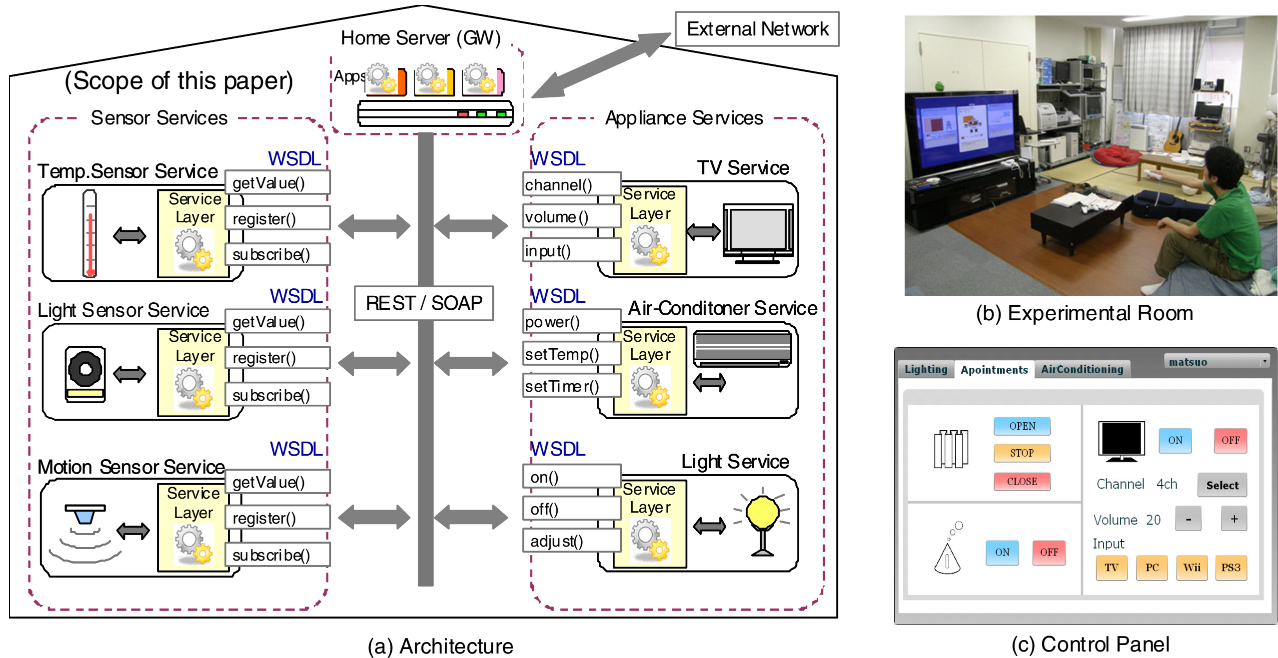


Figure 1. Service-Oriented Home Network System, CS27-HNS

II. Preliminaries

A. CS27-HNS: Service-Oriented Home Network System

We have been developing a service-oriented HNS, called *CS27-HNS*, using actual home appliances [3]. As shown in Figure 1(a), the CS27-HNS consists of *appliance services*, *sensor services*, and a *home server* that manages and controls the services. Every device is abstracted as a service, where features of the device are exhibited as Web services, encapsulating a device-proprietary protocol under the service layer. These services are deployed in our experimental room (Figure 1(b)). A user can control the services with various user interfaces, such as a control panel (Figure 1(c)).

Each appliance service has a set of methods (Web-APIs) that operate vendor-neutral features of the appliances. These methods can be executed by external applications (usually installed in the home server) using standard Web service protocols (i.e., REST or SOAP). For example, a TV service has methods for selecting channels, volume, input sources, etc., which are commonly included in any kinds of TVs. To select channel No.4 of the TV, one can just access a URL `http://cs27-hns/TVService/channel?no=4` with a Web-supported application (e.g., Web browser). The framework for developing the appliance services has been already reported in our previous research [3].

On the other hand, the sensor services in Figure 1(a) are our new challenge and the main scope of this paper. By our design, every sensor service in the CS27-HNS will have the same set of Web-APIs. The method `getValue()` is for retrieving the current normalized value of the sensor reading. Other methods `register()` and `subscribe()` are for the context-aware services in the HNS. The details of the APIs will be presented in Section III.

B. Context-Aware Services in HNS

By introducing sensors in the HNS, it is possible to gather various *contexts* [6] of the environment. We assume that every sensor has a single *property*, which is a variable storing the current value measured by the sensor. [1] A context can be defined as a *condition* over the sensor property. A context can be used for triggering the appliance services, which implements a *context-aware service* in the HNS.

For example, suppose that a temperature sensor `t` has a property `temperature`. One may define a context `Hot` by a condition "the room temperature is 28°C or greater", denoted by `[Hot: t.temperature>=28]`. Then, binding `Hot` to an API `AirConditioner.cooling()` achieves a context-aware air-conditioning service.

In the conventional HNS, such context-aware services have been implemented as proprietary applications, where the sensors and appliances were tightly coupled. In the above example, the air conditioner and its dedicated temperature sensor were controlled by a proprietary program logic. It was basically impossible to replace the sensor with a different one, or to reuse the sensor with other appliances.

The sensor-as-a-service concept [7][8] can de-couple the sensors from the appliances, which improves flexibility and

[1] There may exist a sophisticated sensor that can measure multiple properties at a time. In such a case, we virtually divide the sensor into pieces, each of which forms a sensor with a single property.

reusability of the sensor devices. However, no systematic method for adapting a given sensor device to the SOA has been proposed, as far as we know. Thus, the features of the sensor services vary from project to project, even within the domain of the HNS. These facts impose considerable development effort of the sensor services.

C. Research Goal and Approach

Our research goal is to reduce the development effort of the sensor services, especially within the context of the HNS. To achieve the goal, we develop the following three technologies in this paper.

- **Sensor Service Framework (SSF):** supports developers to adapt sensor devices to the SOA.
- **Sensor Mashup Platform (SMuP):** allows dynamic composition of the existing sensor services.
- **Sensor Service Binder (SSB):** helps non-expert developers create context-aware services within the HNS.

III. Sensor Service Framework: Application Framework for Sensor-as-a-Service

A. Sensor Service Framework (SSF)

The key idea to reduce the development effort is to identify "typical tasks" requested for all kinds of sensor devices, and to delegate the tasks to a common framework. The framework is called the *sensor service framework (SSF)*.

Regardless of the type of sensors, a sensor in the HNS is typically used either to retrieve environmental data, or to check pre-defined contexts. To support these typical tasks, the proposed SSF prescribes the following two services for every sensor device.

- **Normalized Polling Service:** returns the current value of the sensor property in a device-neutral form.
- **Context Management Service:** monitors registered contexts, and publishes an event notification.

B. Normalized Polling Service

The polling service allows client applications to actively retrieve the data from the sensor device. Our SSF prescribes `getValue()` method as an API of the polling service.

In general, a sensor device produces raw data that requires device-specific interpretation. Two different types of temperature sensors produce different raw data as sensor readings. For each device, there exists a method (often given as a formula) that *normalizes* the raw data into a meaningful property. For example, Phidgets temperature sensor (#1124 - Precision Temperature Sensor) [14] has a formula that normalizes a sensor value to a temperature in Celsius.

```
Temperature=(SensorValue x 0.2222)-61.111
```

Preferably, such device-specific normalization should be encapsulated under the sensor service, so that the client can use the data without deep insight of the device. Thus, we want the "normalized" polling service.

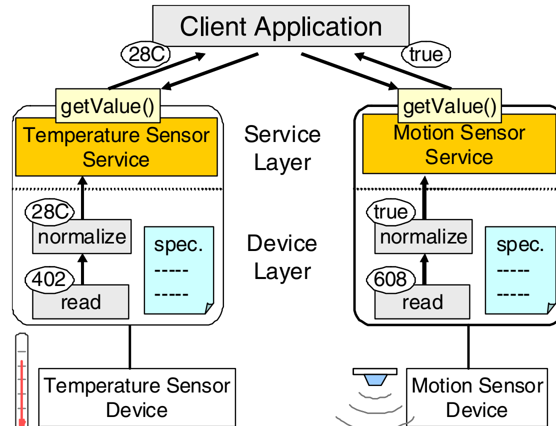


Figure 2. Normalized Polling Service of the SSF

To achieve the normalized polling service, the SSF specifies the sensor service by two layers: the *device layer* and the *service layer*. The device layer involves the device-specific methods to read the raw data and to normalize the data. On the other hand, the service layer provides the device-neutral method `getValue()`, which returns the normalized sensor data. The service layer is supposed to be exhibited as a Web service, to achieve the platform-independent access.

Figure 2 shows the normalized polling services of two sensor devices. The left side shows an example of a temperature sensor. When `getValue()` is executed, the device layer reads the current sensor value 402 from the device. The data is then normalized into 28°C, and delivered to the service layer. Finally, the service layer returns 28°C as the return value of `getValue()`. The right side shows an example of a motion sensor. In this example, the analog value of the motion sensor is normalized into a Boolean value, where true and false represent that a motion is detected or not, respectively.

As supplementary information, the SSF allows the device layer to involve the *specification* of the sensor, describing [sensor id, sensor type, property name, unit, data type, data range, location, description]. The specification is meta-data that helps the clients understand the sensor data, and is obtained by `getSpecification()` method. For example, a specification of a temperature sensor is like

```
[tempSensor001, Temperature, temperature,
 celcius, int, [-30..80], "living room",
 "Phidgets 1124 Precision Temperature"]
```

C. Context Management Service

In the HNS, a sensor is typically used for checking a context, which is defined by a condition over the sensor property. In many applications, such contexts have been managed in the client side. For instance, recall the context `[Hot: t.temperature >= 28]` in Section II-B. In the conventional approach, the client periodically polls the

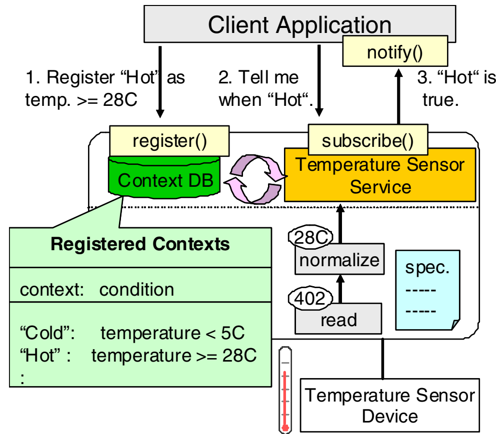


Figure 3. Context Management Service of the SSF

temperature sensor `t` via `getValue()`, and evaluates the condition of `Hot`. Since the same kind of context evaluation is repeatedly done in many applications, this approach increases not only the traffic between the sensor and the clients, but also the complexity of the clients,

Our SSF delegates the task of the context management to the sensor service. Figure 3 shows an example where a client application delegates the management of `Hot` to a temperature sensor service. The client first registers the context using `register()` method. The context is given by its *name* and a *condition*. The context name is a unique label identifying the context, whereas the condition is an expression composed of a sensor property and comparison operators (`== != > < >= <=`). Then, using `subscribe()` method, the client tells the service a *callback address* to be notified when the context is satisfied. The callback address can be given as a URL of any Web service. The service keeps monitoring the sensor value. When any registered context becomes true, the service invokes the corresponding Web service. Note that for a single sensor service, different clients can register own contexts, and that any registered context can be reused by multiple clients.

The SSF also prescribes other supplementary methods, including `getRegisteredContexts()` to obtain registered contexts, `getSubscriptions()` to list the current subscriptions, `pause()` and `resume()` to pause and resume a given subscription.

D. Implementing Sensor Services with SSF

We have implemented the proposed SSF as a Java class library. Figure 4 shows an UML class diagram of the library (only the essential portions are shown).

The SSF specifies classes in the right-hand side of the figure. The classes are divided into the device and service layers. The device layer contains an abstract class, `SensorDevice`, specifying the three methods described in Section III-B. These methods are supposed to be implemented in each concrete sensor class. A sensor device has a `Specification` storing the meta-data of the device.

In the service layer, `SensorService` has a sensor device from which a normalized sensor value is obtained (via `getValue()`). Upon the execution of `register()` and `susbscribe()`, the sensor service creates a `Context` and a `Subscription`, respectively. A `ContextMonitor` periodically evaluate the registered contexts, and notifies the corresponding clients when a context becomes true.

The left side of Figure 4 shows classes for concrete sensor services and devices, which are to be implemented by developers themselves. For a given sensor device, a developer has to create two new classes: a *concrete sensor* class and a *concrete service* class. The concrete sensor class must implement `SensorDevice` of the SSF, using the device-specific operations. Individual sensor vendors typically distribute manuals and/or SDK, providing sufficient knowledge of how to read and interpret the sensor value in a program. Using the information, the developer writes code for `read()` and `normalize()` methods.

Implementing the concrete service class is simple. The developer writes code so that the service class extends `SensorService` of the SSF, and wraps a concrete sensor object via `device` attribute of `SensorService`. Then, the developer deploys the service class as a Web service using a preferred middleware such as Apache Axis.

We have observed in many practical cases that the development effort for creating the above two classes was quite small. For instance, our implementation of `TemperatureSensor` with Phidgets #1124 device comprises just 43 lines of Java code, and `Temperature SensorService` does only 17 lines of code. A average-trained student could program them less than one hour. More detailed evaluation will be conducted in Section VI.

E. Using Sensor as a Service in HNS

Here we give example scenarios of using a sensor service. Suppose that we have `TemperatureSensorService` deployed as a REST Web service. To obtain the current temperature, a client just invokes the following URL.

```
http://hns/TemperatureSensorService/getValue
```

Let us see how the context-aware air-conditioning service in Section II-B can be implemented easily. The service can be created by the following sequence of REST invocations.

(1) Register a context `[Hot:temperature>=28]`.

```
http://hns/TemperatureSensorService/register?
context=Hot&condition='temperature>=28'
```

(2) Bind `Hot` to `AirConditioner.cooling()`.

```
http://hns/TemperatureSensorService/subscribe?
context=Hot&notify='http://hns/AirConditioner
Service/cooling'
```

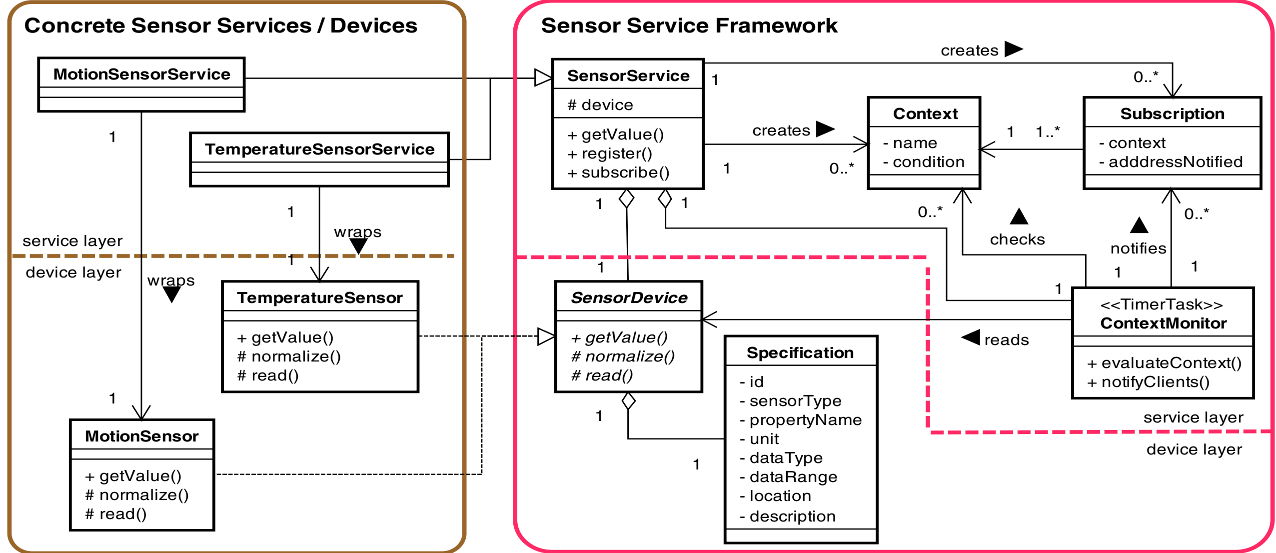


Figure 4. UML Class Diagram of Sensor Service Framework

IV. Sensor Mashup Platform: Composition Platform for Sensor-as-a-Service

A. Sensor Mashup Platform (SMuP)

We propose the *sensor mashup platform (SMuP)*, adding the "composition-as-a-service" concept [13] to the SSF.

Basically, a sensor service with the SSF can manage simple contexts over a single sensor property. For example, the temperature sensor service in Figure 3 can manage `Hot` and `Cold` which are defined over the property `temperature`.

However, using multiple sensor services together produces a valuable property characterizing more sophisticated contexts. For example, using the temperature sensor together with a humidity sensor, we can compute the *discomfort index (DI)* [15], characterizing comfortability of air-conditioning. Another example is that using two different light sensors can measure the average brightness level of the room.

The SMuP allows the developers the *sensor mashup*, which dynamically creates such composite sensors from the existing sensor services.

B. Creating Composite Sensor Services with SMuP

Unlike the SSF, the SMuP is a Web service already deployed in the HNS, on which the developers can create *virtual composite sensors* online. We use the term "virtual" to refer to a sensor that does not contain an actual sensor device. Instead, the sensor property is derived from other sensor services, not from the sensor device. For this, the SMuP provides three APIs: `importSensorService()`, `createSensor()`, `addProperty()`.

The method `importSensorService()` takes two parameters: an *URL* of the sensor service and a *reference label*. This method makes an existing sensor service available within the SMuP, and the imported service can be referred by the specified label. The method `createSensor()` creates a new virtual sensor in the SMuP, by taking a *sensor name* as a parameter. The method `addProperty()` takes three parameters: a *sensor name*, a *property name* and a *formula*. The method defines a new property as a formula over the reference labels, and adds the property to the virtual sensor. In the formula, arithmetic operators (`+ - * / %`), comparison operators (`== != > < >= <=`) and logical operators (`&& || !`) can be used to mash up the properties.

Figure 5 illustrates a workflow of creating a new sensor service, `DiscomfortIndexSensorService` from two existing services `TemperatureSensorService` and `HumiditySensorService`. The workflow consists of three steps, where the discomfort index (DI) is computed from a temperature (`t`) and humidity (`h`) as follows:

```
DI := 0.81*t + 0.01*h*(0.99*t-14.3) + 46.3
```

(Step 1) The client application imports the temperature and humidity sensor services with labels `t` and `h`, respectively, using `importSensorService()`.
(Step 2) The client executes `createSensor()` to create the discomfort index sensor.
(Step 3) The client executes `addProperty()` to add property `DI` with the above formula, to the created sensor.

Similarly, one can easily create the average brightness sensor service using two different light sensors, as shown in the right side of Figure 5.

C. Using Composite Sensors Created by SMuP

Since every virtual sensor created by the SMuP conforms to the SSF, the sensor can be used analogous to the ordinary sensor services. That is, the methods `getValue()`,

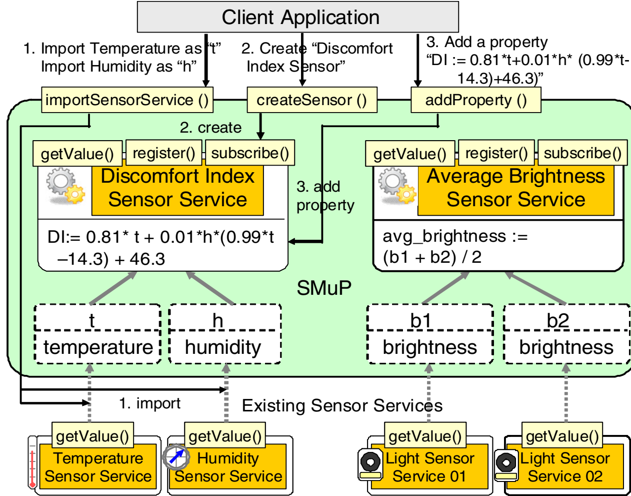


Figure 5. Creating Composite Sensor Services with the SMuP

`register()` and `subscribe()` can be used in each virtual sensor. When a client executes `getValue()` for a virtual sensor, the virtual sensor first obtains the current values of the "child" sensor services. Based on the values, the formula is evaluated as a return value of the API.

For example, suppose that a client executes `getValue ()` of the discomfort index sensor in Figure 5. The sensor first obtains values of `t` and `h` from the temperature and humidity sensor services, respectively. Suppose that `t=24` (°C) and `h=35` (%) are obtained. Then, the formula of `DI` is evaluated and the value `69.051` is returned.

Also, one can register the context to the virtual sensor. For example, according to the statistics [15], people feel the air-conditioning is *comfortable* when DI is between 65 and 70. So, we can register the context `[comfortable: 65<=DI && DI<70]` to the discomfort index sensor via `register()`. Then, any client can bind the context to an HNS operation by `subscribe()`. The sensor service keeps monitoring `comfortable`, and notifies the corresponding addresses when the context becomes true.

V. Sensor Service Binder: User-Friendly Interface for Context-Aware Service Creation

A. Sensor Service Binder (SSB)

The proposed SSF and SMuP facilitates the development of sensor services in the HNS. However, it is yet challenging for non-expert users to create the context-aware services using the sensor services. To cope with this problem, we have developed a novel service creation environment, called *Sensor Service Binder (SSB)*, in our CS27-HNS.

The SSB provides a graphical user interface for rapid creation of context-aware services, which acts as a front-end of the SSF. The SSB automatically parses the WSDL of the sensor/appliance services within the HNS. It then displays the information in an intuitive and user-friendly form. The user can play with the services through basic widgets such as buttons, lists and textboxes. Since the SSB requires no expertise of Web services, it can minimize the careless faults in operating the sensor services. Also, the SSB can list all contexts and callback APIs registered in all sensor services. This allows users to overlook the entire list of available contexts and corresponding services.

The SSB provides the following two primary features supporting end-users.

- **(Context Registration Feature)** Register a user-defined context by executing `register()` method of the sensor service.
- **(Context Subscription Feature)** Bind a registered context to a Web-API of an HNS appliance using `subscribe()` method.

B. Context Registration Feature of SSB

Figure 6(a) shows a screenshot of the registration feature. The left side of the screen is the registration pane. A user first chooses a desired sensor service from the drop-down list, and then enters a context name in the textbox. In the below of the textbox, a sensor property is automatically shown. The user defines a context condition by an expression over the property. In the default mode, the SSB allows only a constant value and a comparison operator. Finally, the user presses the "Register" button. The SSB registers the context to the service by invoking `register()` method.

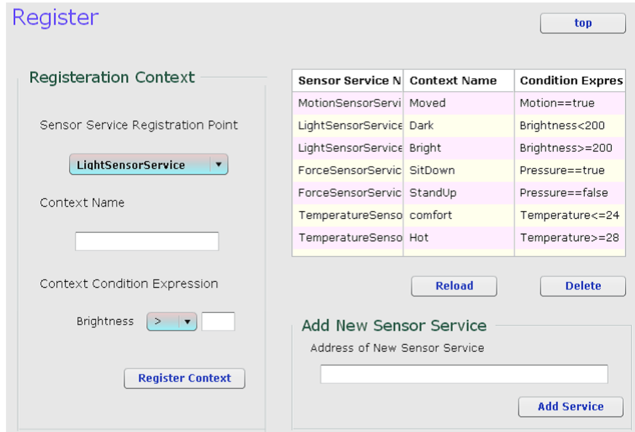
The right side of the screen represents a list of contexts that were already registered. The list is dynamically created by `getRegisteredContexts()` method of the SSF. Each line contains a context name, a context condition and a sensor service where the context is registered. The user can check if the created context is registered. The user can also discard unnecessary contexts by just pressing "Delete" button. The SSB requests the service to delete the context.

C. Context Subscription Feature of SSB

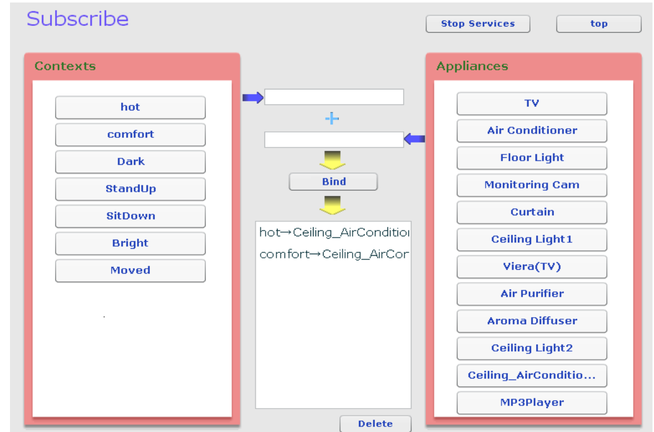
This feature helps a user bind a registered context to a Web-API of the appliance operation. Figure 6(b) shows a screenshot of the context subscription feature. The left side of the screen enumerates the registered contexts, each of which is labeled by the context name. When a user clicks a preferred context, the context is chosen for the binding.

The right side of the screen shows the list of appliance services deployed in the HNS. When a user clicks a preferred appliance, a menu of operations of the appliance is popped up. Then the user chooses an operation to bind. The list of appliances and the menu of operations are automatically generated by parsing the WSDL of the appliance services.

Finally, when the user clicks "Bind" button in the center, the SSB subscribes the binding by executing `subscribe()` method. This completes a service creation. The subscribed contexts are shown in the textbox in the center, where the user can delete any binding.



(a) Context Registration



(b) Context Subscription

Figure 6. Screenshots of Sensor Service Binder

VI. Evaluation

A. Practical Feasibility

Using the SSF, we implemented 16 sensor services, and deployed them in the CS27-HNS with Apache Axis2. As summarized in Table I, most sensor services wraped Phidgets devices. The total lines of code (LOC) for adapting Phidgets #1114 temperature sensor was just 65. Implementing the same temperature sensor service without the SSF comprised 4,004 LOC. Thus, we can see how the SSF can reduce the development effort, significantly.

The SSF was well applied to not only Phidgets sensors but also other devices like Panasonic Lifinity and Weather Goose. More importantly, the SSF could be used to wrap the weather news as a sensor service. As long as the developer can implement `read()` and `normalize()` methods, the SSF can adapt any data source, not limited to a sensor device. Thus, the SSF can be applied to a wide range of applications, such as a database, the appliance status and information resources on the Web (news, stock, blog).

Although the SSF supports only two primitive services (see Section III), the services were enough to implement many practical context-aware services in the CS27-HNS. They include the air-conditioning with the discomfort index, the automatic light control, the energy peak control, the all shutdown when leaving, the couch potato prevention, etc.

We have also conducted an experiment of creating context-aware services using the 16 sensor services and the SSB. The total 6 subjects participated in the experiment. None of the subjects was familiar with the HNS or the SSF. Using the SSB, each subject performed the registration of 5 contexts (Task T1), and the subscription of 5 services (Task T2). It was shown that the average time taken for Tasks T1 and T2 are 76 seconds and 74 seconds, respectively. This result indicates the efficiency of the SSB and SSF in that non-expert developers could build the context-aware services from scratch, within just a few minutes.

B. Validity of Approach

One may doubt the validity of our approach wrapping a sensor with the expensive Web service. Indeed, many existing studies on the sensor networks often assume environments with severe requirements on power, size, network connectivity, mobility, scalability, etc. Compared to those, the HNS is a physically mild environment, since the sensor devices are fixed on the house where the power and the network are guaranteed. Thus, in the domain of the HNS, we put more weight on the reliability and interoperability, which justifies the sensor-as-a-service approach.

C. Related Work

Several studies on the service-oriented middleware for the sensors and context-aware applications have been reported, for example, SOCAM [10], Sens-ation [11], Atlas [12]. These middleware systems give much weight to high-level context management, such as context interpretation, reasoning, discovery, etc. However, they do not give the details of how to adapt the physical sensors to the middleware. Therefore, our SSF can be used to complement these middleware systems, in implementing the concrete device adapter as sensor service.

Concerning the proposed SSB, there are studies for the user interface for building the context-aware services. Sheng et al. [9] presented a graphical user interface for modeling context-aware application using UML. Dey et al. [16] presented aCAPpella, where a user can program contexts by demonstration. Compared to these methods, our SSB provides a less-expressive but light-weight approach for non-expert users, limiting the contexts to the ones managed by the SSF. Thus, in exchange for the limitation, the users can easily perform “scrap and build” of context-aware services within a couple of minutes.

Table I
DEVELOPED SENSOR SERVICES IN CS27-HNS

Sensor Service	LOC	Wrapped Device or Data Source
Temperature	65	Phidgets #1114
Humidity	87	Phidgets #1107
Light	60	Phidgets #1105
Force	67	Phidgets #1106
GasPressure	65	Phidgets #1126
Sound	68	Phidgets #1133
Motion	84	Phidgets #1111
Rotation	74	Phidgets #1109
Contact	74	Phidgets #3560
OutHumidity	65	Phidgets #1125
OutLight	65	Phidgets #1127
OutTemperature	60	Phidgets #1124
PeopleCounter	90	Phidgets #1023 RFID + Custom WS
PowerConsumption	71	Panasonic Lifinity
WeatherGoose	148	Weather Goose II
WeatherNews	141	Livedoor Weather Hack RSS

VII. CONCLUSION

To facilitate the development of "sensor as a service" in the home network system (HNS), we have proposed three novel technologies in this paper: (1) *Sensor Service Framework (SSF)*: application framework supporting developers to adapt sensor devices to the SOA, (2) *Sensor Mashup Platform (SMuP)*: platform for dynamic composition of the sensor services, and (3) *Sensor Service Binder (SSB)*: user-friendly interface helping non-expert developers create context-aware services within the HNS. These technologies have been implemented on an actual home network, CS27-HNS, to demonstrate the practical feasibility.

As for the future work, we are currently extending the SMuP to be able to manage *timing constraints* among contexts. We are also implementing several extensions of the SSB. One is the discovery feature, with which users can search sensors and appliances by name, location, purpose, etc. Another issue is to share and reuse the existing contexts, facilitating the context creation and registration.

ACKNOWLEDGMENT

This research was partially supported by the Japan Ministry of Education, Science, Sports, and Culture, Grant-in-Aid for Young Scientists (B) (No.21700077) and Research Activity Start-up (No.22800042).

REFERENCES

[1] C. L. Wu, C. F. Liao, and L. C. Fu, "Service-oriented smart home architecture based on osgi and mobile agent technology," in *IEEE Trans. on Systems, Man, and Cybernetics (SMC), Part C*, vol. 37, no. 2, 2007, pp. 193–205.

[2] J. Bourcier, A. Chazalet, M. Desertot, C. Escoffier, and C. Marin, "A dynamic-soa home control gateway," in *Proc. the 3rd IEEE International Conference on Services Computing (SCC)*, 2006, pp. 463–470.

[3] M. Nakamura, A. Tanaka, H. Igaki, and K. Matsumoto, "Constructing home network systems and integrated services using legacy home appliances and web services," *Int'l J. of Web Services Research*, vol. 5, no. 1, pp. 82–98, 2008.

[4] H. Igaki, H. Seto, M. Fukuda, and M. Nakamura, "Mashing up multiple logs in home network system for promoting energy-saving behavior," in *Proc. of 8th Asia-Pacific Symposium on Information and Telecommunication Technologies (APSITT2010)*, vol. CDROM, June 2010.

[5] M. Nakamura, H. Igaki, Y. Yoshimura, and K. Ikegami, "Considering online feature interaction detection and resolution for integrated services in home network system," in *the 10th Int'l Conf. on Feature Interactions in Telecommunications and Software Systems (ICFI2009)*, 2009, pp. 191 – 206.

[6] A. K. Dey and G. D. Abowd, "Towards a better understanding of context and context-awareness," in *Proc. the 1st International Symposium on Handheld and Ubiquitous Computing (HUC)*, 1999, pp. 304–307.

[7] M. G. Lozano, P. Horling, F. Moradi, and E. Tjornhammar, "Supporting c2 with a service oriented framework for opportunistic sensors and sensor networks," in *Proc. International Command and Control Research and Technology Symposium (ICCRTS2009)*, 2009.

[8] S. Alam, M. M. R. Chowdhury, and J. N. Muhl, "Senaas: An event-driven sensor virtualization approach for internet of things cloud," in *Proc. International Conference on Networked Embedded Systems for Enterprise Applications (NESEA2010)*, 2010, pp. 1–6.

[9] Q. Z. Sheng, S. Pohlenz, J. Yu, H. S. Wong, A. H. Ngu, and Z. Maamar, "Contextserv: A platform for rapid and flexible development of context-aware web services," *Proc. the 31st International Conference on Software Engineering (ICSE)*, pp. 619 – 622, 2009.

[10] T. Gua, H. K. Punga, and D. Q. Zhang, "A service-oriented middleware for building context-aware services," *Journal of Network and Computer Applications*, vol. 28, pp. 1 – 18, 2005.

[11] T. Gross, T. Egla, and N. Marquardt, "Sens-ation: A service-oriented platform for developing sensor-based infrastructures," *International Journal of Internet Protocol Technology (IJIPT)*, vol. 1, no. 3, pp. 159–167, 2006.

[12] J. King, R. Bose, H. i Yang, S. Pickles, and A. Helal, "Atlas: A service-oriented sensor platform hardware and middleware to enable programmable pervasive spaces," in *Proc. the 31st IEEE Conference on Local Computer Networks (LCN)*, 2006, pp. 630–638.

[13] M. B. Blake, W. Tan, and F. Rosenberg, "Composition as a service," *IEEE Internet Computing*, vol. 14, no. 1, pp. 78–82, 2010.

[14] *Phidgets - Products for USB Sensing and Control*, http://www.phidgets.com/.

[15] J. R. Bosen, "Discomfort index," in *Reference data section, Air Conditioning, Heating, and Ventilating*, 1959.

[16] A. K. Dey, R. Hamid, C. BeckMann, I. Li, and D. Hsu, "a cappella: Programming by demonstration of context-aware applications," *Proc. CHI*, vol. 6, no. 1, pp. 33 – 40, 2004.