Roger Lee (Ed.)

Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing 2012
Supporting End-User Development of Context-Aware Services in Home Network System

Masahide Nakamura, Shuhei Matsuc, and Shinsuke Matsumoto

Abstract. The home network system (HNS, for short) provides value-added and context-aware services for home users, by orchestrating networked home appliances and sensors. Although the HNS services have been developed by system vendors, there exist strong needs that the end-users create their own services according to their convenience. This paper presents a novel service creation environment, called Sensor Service Binder (SSB), which provides a user-friendly interface for creating context-aware services within the HNS. Built on top of the service-oriented HNS, the SSB allows non-expert users to register contexts using the sensors, and to bind the registered context to any operation of the networked appliances. Experimental evaluation with an actual HNS showed that the effort for service creation was reduced to 10% by introducing the proposed SSB.

Keywords: home network system, home appliances, sensors, context-aware services, end-user development.

1 Introduction

Research and development of the home network system (HNS, for short) is recently a hot topic in the area of ubiquitous computing applications [13][1]. Orchestrating house-hold appliances and sensors via network, the HNS provides various value-added services for home users. Typical services include; the remote control service inside/outside home, the monitoring service of device and environment status, and the integrated service of multiple appliances [9].

Masahide Nakamura · Shuhei Matsuo · Shinsuke Matsumoto
Graduate School of System Informatics, Kobe University, 1-1 Rokkodai-cho, Nada-ku, Kobe, Hyogo 657-8501, Japan
e-mail: {masa-n, shinsuke}@cs.kobe-u.ac.jp, matsu@ws.cs.kobe-u.ac.jp

springerlink.com © Springer-Verlag Berlin Heidelberg 2013
In this paper, we especially focus on the context-aware service [10] within the HNS, which automatically triggers service operations based on contextual information. A context-aware service is basically implemented by binding a context to an operation of the HNS device. The context is usually characterized by device states and/or environment properties gathered by sensors. For instance, binding a context “Hot: temperature>=28C” with an operation “airConditioner.cooling()” implements an autonomous air-conditioning service.

Such context-aware services have been developed by vendors in a “ready-made” form. However, due to the variety of user’s tastes on the context, appliances deployed and surrounding environment, the conventional ready-made services do not necessarily cover all requirements of end-users. In the above example, a user may feel hot when the temperature is 26C. Also, she may want to use a fan instead of the air-conditioner, as she dislikes the air-conditioner.

To cope with such fine requirements by end-users, we propose a context-aware service creation environment for the HNS, called Sensor Service Binder (SSB). We have previously developed the service-oriented HNS [9], where operations of the HNS appliances and sensors are exhibited as Web services. The SSB is built on top of the service-oriented HNS, and supports end-users to perform the following two primary tasks to create a context-aware service:

- **Register**: Define an end-user context with device states and sensor values, and register it to the server.
- **Subscribe**: Bind a registered context to a desired appliance operation. The operation is triggered when the context becomes true.

We have conducted an experimental evaluation where non-expert users create simple context-aware services with the SSB. It was shown that the time taken for each subject to create a service was a couple of minutes. We also observed that the number of faults in invoking Web service APIs was significantly reduced. These facts imply that the proposed SSB can contribute to efficient and reliable end-user development of context-aware services in the HNS.

2 Related Work

There have been several methods and systems that facilitate development of context-aware applications. In [4] and [12], middleware supports for context-aware applications development have been proposed. Using the middleware, developers can create own applications rapidly, by combining building blocks for reading sensor values, reasoning contexts, executing callbacks, etc. Also, [11] presented a graphical user interface for modeling context-aware application using UML. These studies are supposed to be provided, but are different from our SSB.

In [3], a user interface, which end-users was proposed. User instruction. The demonstration manually to bind scenes with. Our SSB provides a more controls to the ones constructed are created by a set of simple do “scrap and build” of appliance.

3 Preliminaries

3.1 Service-Oriented

Applying the service-oriented HNS devices. Wrapping protocol loose-coupling and platform uses the devices. Several smart home appliances [13][1] and.

In our previous work [9], an service-oriented HNS, called C27, shown in Figure 1, the C27 and a home server that manages device is abstracted as a set of services (Web-APIs), encapsulation.

Each appliance service exposes features of the appliance applications (usually installed protocols [i.e., REST or SOAP], selecting channels, volume, in any kinds of TVs. To select http://c27-hns/TV application (e.g., Web browser).

On the other hand, every of Web-APIs. The API get an environment property, set register() and subscribe explained in the next section.
are supposed to be provided for sophisticated programmers and designers. So they are different from our SSB helping development by non-expert users at home.

In [3], a user interface, aCAPpella, for context-aware application development by end-users was proposed. Using aCAPpella, a user can program contexts by demonstration. The demonstration is captured by a camera and sensors. Then it is annotated manually to bind scenes with relevant contexts.

Our SSB provides a more light-weight approach for home users, by limiting contexts to the ones constructed by sensor services within the HNS. Since an application is created by a set of simple IF-THEN rules of ready-made services, users can easily do “scrap and build” of applications within a couple of minutes.

3 Preliminaries

3.1 Service-Oriented HNS

Applying the *service-oriented architecture (SOA)* to the HNS is a smart solution to achieve the programmatic interoperability among heterogeneous and distributed HNS devices. Wrapping proprietary control protocols by Web services provides loose-coupling and platform-independent access methods for external software that uses the devices. Several studies have been reported on the service orientation of home appliances [13][1] and sensor networks [5][6].

In our previous work [9][8], we have also designed and implemented a service-oriented HNS, called CS27-HNS, using actual home appliances and sensors. As shown in Figure 1, the CS27-HNS consists of *appliance services*, *sensor services*, and a *home server* that manages and controls the services. Each appliance (or sensor) device is abstracted as a service, where features of the device are exhibited as Web services (Web-APIs), encapsulating a device-proprietary protocol under the service layer.

Each appliance service has a set of Web-APIs that operate vendor-neutral features of the appliances. These Web-APIs 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/TVServe/channel?no=4 with a Web-supported application (e.g., Web browser).

On the other hand, every sensor service in the CS27-HNS has the same set of Web-APIs. The API `getValue()` returns the current normalized value of an environment property, such as temperature (C), brightness (lux). Other APIs `register()` and `subscribe()` are for sensor-driven context-aware services, explained in the next section.
3.2 Context-Aware Services in HNS

By using the sensor services in the HNS, it is possible to gather various contexts [2] of the environment and users. A context can be used for triggering appliance services, which implements a context-aware service in the HNS. For example, one may define a context "Hot" to trigger the activation of an air-conditioner service.

Then, binding Hot to a Web-API is possible by using an SSF.

To facilitate the context binding, a special framework is used. When the sensor device reports a temperature sensor value, then the client of a sensor service fires a context-aware service. Note that different clients can subscribe to a same context and that any registered context can be fired.

Using the SSF in the CS27-HNS is easily implemented by the following steps:

1. Define a context (i.e., Hot) and subscribe to the temperature sensor service.

   ```
   http://cs27-hns/context=Hot
   ```

2. Bind Hot to Web-API APIs.

   ```
   http://cs27-hns/consumer?context=Hot
   ```

3.3 End-User Development

Although the CS27-HNS provides context-aware services, it is yet quite challenging (pragmatic with Web service) to develop services. A user needs to write a Web-API, usually described in Web service spec., registered contexts, and write an implementation code for non-expert users to understand and use.

Under this situation, our service platform supports the development of their own services. For this, we propose the Sensor Service Binder (SSB) as a tool that supports a platform-independent service specification and service implementation for a given context, with the context-aware service development, and service deployment.
define a context “Hot” to represent that “the room temperature is 28°C or greater”. Then, binding Hot to a Web-API AirConditioner.cooling() achieves a context-aware air-conditioning service.

To facilitate the context management, the sensor services in our CS27-HNS conform to a special framework, called Sensor Service Framework (SSF) [8]. For every sensor device, the SSF provides autonomous monitoring/notification services for the device, performed by a pair of Web-APIs: register() and subscribe(). A client of a sensor service first defines a context by a logical expression over the sensor value. Then, the client registers the context to the service using register() method. Next, the client executes subscribe() to tell a callback Web-API. After that, the service keeps monitoring the sensor value. When the registered context (i.e., the logical expression) becomes true, the SSF invokes the callback Web-API. Note that different clients can register multiple contexts in the same sensor service, and that any registered context can be shared and reused by different subscriptions.

Using the SSF in the CS27-HNS, the example air-conditioning service can be easily implemented by the following sequence of REST invocations.

1. Define a context Hot as an expression “temperature $\geq 28$”, and register it to TemperatureSensorService.

```
http://cs27-hns/ TemperatureSensorService/register?
context=Hot & expression=’temperature>=28’
```

2. Bind Hot to Web-API AirConditioner.cooling().

```
http://cs27-hns/ TemperatureSensorService/
subscribe?context=Hot & notify=’http://cs27-hns/
AirConditionerService/cooling’
```

### 3.3 End-User Development of Context-Aware Services

Although the CS27-HNS with the SSF facilitates the development of context-aware services, it is yet quite challenging for end-users, who have no expertise in programming with Web services, to develop their own services. To use a sensor (or appliance) service, a user has to understand the interface and end point of the Web-API, usually described in WSDL. Also, the information managed by the SSF (sensor spec., registered contexts, callback APIs, etc.) are all described in XML. It is hard for non-expert users to understand and use the sensor services correctly.

Under this situation, our objective is to support the non-expert end-users to create their own services. For this, we propose a novel service creation environment, called Sensor Service Binder (SSB), built on top of the CS27-HNS.
4 Sensor Service Binder: User-Friendly Interface for Context-Aware Service Creation

4.1 Overview

The SSB provides a graphical user interface for rapid creation of context-aware services, which acts as a front-end of the CS27-HNS with the SSF. The SSB automatically parses the WSDL and the XML files of the sensor/appliance services. 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, without knowing underlying information like the service end point, the message types, etc. Since the SSB restricts the user’s input from the GUI only, it is possible to minimize the careless faults in operating services.

Also, the SSB can search and aggregate contexts and callback Web-APIs registered in all sensor services. This feature allows users to overlook the entire list of available contexts and corresponding services. The list can be used to verify, reuse and refine the existing context-aware services, which were difficult activities by the SSF only.

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

- **(F1: Registration Feature)** Register a user-defined context by executing `register()` method of the sensor service.
- **(F2: Subscription Feature)** Bind a registered context to a Web-API of a HNS operation using `subscribe()` method of the sensor service.

4.2 Context Registration Feature of SSB

This feature allows a user to define and register a context using the sensor services. In the SSB, a context is defined by a `name` and a `condition`. The context name is a unique label identifying the context, whereas the context condition is a logical expression composed of sensor values and comparison operators.

Figure 2(a) shows a screenshot of the registration feature. The left side of the screen is the registration pane. A user first chooses a favorite sensor service from the drop-down list, and then enters a context name in the textbox. In the below of the textbox, an attribute of the sensor service is automatically derived and shown. The user defines a context condition by an expression over the attribute. In the default mode, the SSB allows only a constant value and a comparison operator, just for convenience. 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
Supporting End-User Development of Context-Aware Services

(a) Screen of Context Registration

(b) Screen of Context Subscription

Fig. 2 Screenshot of Sensor Service Binder
registered. The user can also discard unnecessary contexts by just pressing “Delete” button. The SSB requests the service to delete the context.

4.3 Context Subscription Feature of SSB

This feature helps a user bind a registered context to a Web-API of the appliance operation. Figure 2(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.

4.4 Example

As an illustrative example, let us create a simple service, say automatic TV service with the SSB. This service turns on a TV only when a user sits down on a couch. We suppose that a Force sensor is deployed under the couch to detect a human sitting on the couch.

First, we define and register a context SitDown using the registration feature. From the drop-down list of sensors (see Figure 2(a)), we choose the Force sensor. Then we enter the name SitDown in the textbox, and make a condition as pressure==true. Finally, we press the register button.

Next, we bind SitDown to TV.on() using the subscription feature. We first choose SitDown from the context list of Figure 2(b). Then, we choose TV from the appliance list, and on from the operation menu. Finally, we press the bind button. Similarly, we create a context StandUp as pressure==false, and then bind it to TV.off(), which completes the creation of the automatic TV service.

5 Evaluation

5.1 Experiment Setting

To evaluate the effectiveness, we have conducted an experiment of service creation with (and without) the proposed SSB. The total 6 subjects (3 under-graduates, 2 graduates, and 1 faculty) participated. Each subject was familiar with the CS27-HNS.

In the experiment, we asked:

- (T1: context registration) Users to register the following contexts:
  1. SitDown: A force sensor detects a user is sitting down.
  2. StandUp: A force sensor detects a user is standing up.
  3. Dark: A light sensor detects the room is dark.
  4. Hot: A temperature sensor detects the room is hot.
  5. Moved: A motion sensor detects a movement.

- (T2: context subscription) Users to subscribe service for each context:
  1. Turn on a TV when a user sits down.
  2. Turn off a TV when a user is standing up.
  3. Turn on a ceiling light when it is dark.
  4. Turn off a gas heater when it is hot.
  5. Close a curtain when a user is standing up.

Each task was performed:

- [with SSB] Each subject used the proposed SSB.
- [without SSB] Each subject used the CS27-HNS.

To avoid the habituation effect, the other half executed the tasks in reverse order.

The usage of the Web browser was not allowed. The most familiar tool for users was the browser. Another option is to use a Web-API.

The experiment was performed in four days:

1. We gave instructions on how to use the Web-APIs.
2. We showed a sample task.
3. Each subject conducted the task.
4. We showed a sample task.
5. Each subject conducted the task.
6. We interviewed the subject about the service creation.

We have measured the time it took to complete the tasks. We also counted the number of errors made by each subject.
Matsuo, and S. Matsumoto

by just pressing “Delete”

Web-API of the appliance
subscription feature. The
of which is labeled
and make a context
services deployed in the
of operations of the appli-
SSB subscribes the
service creation.
user can

say automatic TV service
sits down on a couch. We
detect a human sitting
by the registration feature.
choose the Force sen-
make a condition as a
scription feature. We first
then, we choose TV from the
we press the bind button.
false, and then bind it

Supporting End-User Development of Context-Aware Services

graduates, and 1 faculty) participated in the experiment. None of the subjects was
familiar with the CS27-HNS or the SSB.

In the experiment, we asked the subjects to do the following tasks.

• (T1: context registration) Each subject defines and registers the following 5
contexts.

1. SitDown: A force sensor detects a pressure.
2. StandUp: A force sensor detects no pressure.
3. Dark: A light sensor measures below 200lux.
4. Hot: A temperature sensor shows above 15C.
5. Moved: A motion sensor detects a motion.

• (T2: context subscription) Each subject binds a registered context to an appli-
cance service. Specifically, each subject creates the following 5 bindings.

1. Turn on a TV when SitDown holds.
2. Turn off a TV when StandUp holds.
3. Turn on a ceiling light when Dark holds.
4. Turn on an air conditioner when Hot holds.
5. Close a curtain when Moved holds.

Each task was performed in two ways.

• [with SSB] Each subject uses the SSB.
• [without SSB] Each subject uses a Web browser to directly access the Web-APIs
of the CS27-HNS.

To avoid the habituation effect, the half of the subjects performed [with SSB] first,
and the other half executed [without SSB] first.

The usage of the Web browser in [without SSB] is due to the fact that it is the
most familiar tool for users that can invoke the Web-APIs. In the experiment, the
subjects were instructed to enter URIs of the Web-APIs in the address bar of the
browser. Another option is to train the subjects for writing programs. However, this
is too expensive and is beyond our assumption of “end-users”.

The experiment was performed as follows.

1. We gave instructions of the experiment as well as the usage of the SSB and the
Web-APIs.
2. We showed a sample task of T1 to the subjects.
3. Each subject conducted T1.
4. We showed a sample task of T2 to the subjects.
5. Each subject conducted T2.
6. We interviewed the subjects for the usability of the SSB and the browser-based
service creation.

We have measured the time taken for completing the tasks, to evaluate the efficiency.
We also counted the number of faults in user’s operations as a reliability measure.
5.2 Result

Figure 3 shows a boxplot of the time taken for the subjects to complete each task. It is shown in task T1 that the context registration with the SSB took 76 seconds on the average, which is 12% of the time taken without the SSB. Similarly in task T2, the context subscription with the SSB took only 74 seconds on the average, which is 9% of the time without the SSB. It is also interesting to see that using the SSB all the subjects completed the task as quickly as one minute and plus, which reflects the user-friendly and intuitive design of the SSB. These results show how the SSB improves the efficiency of the end-user development.

Table 1 shows the number of faults made by subjects in each task, summarized according to (a) individual subjects and (b) fault types. Due to a trouble in recording, data for [T1 without SSB] of subject S1 was omitted.

It was surprising to see that no operational fault was made in any task with the SSB. Among the tasks without the SSB, more faults occurred in T2 since the task of subscription is generally more troublesome than registration. Investigating the type of faults, we found that the subjects were likely to mistype very long URIs of the Web-APIs in the browser. It was also seen that the subjects were often at a loss to identify the correct sensors and appliances. These faults were well circumvented by the GUI of the SSB, which reflects the reliability of the service creation.

On the other side of the successful results, we also recognized limitations of the SSB in the subsequent interview. A subject pointed out: “As more and more contexts are registered, the context list of the SSB will become larger. So, it is quite hard for me to search a correct context.” The same thing happens when the number of sensors and appliances is dramatically increased. To cope with this problem, the SSB has to employ efficient search and reuse techniques for the contexts and services, which is left for our future work.

6 Conclusion and Future Works

We have presented a novel approach and development of context-awareness evaluation with non-expert users through HNS. It was shown that the HNS can significantly increase the number of faults within the framework, thereby improving reliability in developing context systems.

As for the future work, we have developed a system for context awareness and SSB. One important issue is the difficulty of identifying sensors and appliances by users and the difficulty of the SSB to identify faults. By addressing these issues, we are constructing a better interface that is more intuitive for users. The feature interaction feature is currently being developed, which is functional to present context information, and it will be available for the entire group of users. We hope to develop a system that is more intuitive and efficient for users.
Supporting End-User Development of Context-Aware Services

Table 1 Experiment Result: number of faults

(a) Number of faults for individual subject

<table>
<thead>
<tr>
<th>Subject ID</th>
<th>w/o SSB</th>
<th>with SSB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T1</td>
<td>T2</td>
</tr>
<tr>
<td>S1</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>S4</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>S5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S6</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Total</td>
<td>6</td>
<td>19</td>
</tr>
<tr>
<td>Average</td>
<td>1.2</td>
<td>3.2</td>
</tr>
</tbody>
</table>

(b) Number of faults with respect to fault types

<table>
<thead>
<tr>
<th>Fault type</th>
<th>w/o SSB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T1</td>
</tr>
<tr>
<td>Wrong URI of sensor service</td>
<td>4</td>
</tr>
<tr>
<td>Wrong URI of appliance operation</td>
<td>0</td>
</tr>
<tr>
<td>Wrong argument of Web service</td>
<td>1</td>
</tr>
<tr>
<td>Registration to wrong sensor service</td>
<td>0</td>
</tr>
<tr>
<td>Subscription to wrong appliance operation</td>
<td>0</td>
</tr>
<tr>
<td>Wrong context name</td>
<td>0</td>
</tr>
<tr>
<td>Wrong context condition</td>
<td>1</td>
</tr>
</tbody>
</table>

6 Conclusion and Future Directions

We have presented a novel environment, called Sensor Service Binder, for end-user development of context-aware services. We have also conducted an experimental evaluation with non-expert users using the practical home network system, CS27-HNS. It was shown that the SSB significantly reduced the development time and the number of faults within the service creation contributed to the efficiency and the reliability in developing context-aware services.

As for the future work, we are currently implementing several extensions of the SSB. One important issue 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. For these issues, we are constructing a registry within the CS27-HNS to manage metadata for locating services and contexts.

The feature interaction [7] problem is also an important problem to be tackled, which is functional conflicts among different services. The feature interactions can occur as well within the services created by the SSB users. For instance, if different users bind incompatible appliance operations with the same context, a race condition occurs, leading to unexpected behaviors. In the future, we plan to develop a validation feature within the SSB, which detects and resolves feature interactions among the user-made services.
Acknowledgements. This research was partially supported by the Japan Ministry of Education, Science, Sports, and Culture [Grant-in-Aid for Scientific Research (C) (No.24500079), Scientific Research (B) (No.23300009)], and Kansai Research Foundation for technology promotion.

References