International Journal of Smart Home
Vol. 9, No. 8 (2015), pp. 151-162
http://dx.doi.org/10.14257/ijsh.2015.9.8.16

A Proposal for Deriving Timing Constraint Context on Using Multiple Sensor Web Servers in Service-Oriented Home Network

BenYan[1], Hua-Ping Yao[1], Masahide Nakamura[2] and Shinsuke Matsumoto[2]

[1]LuoYang Institute of Science and Technology
No. 90 Wangcheng Road, Luolong District,
Luoyang City, Henan Province, 471023, China
[2]Kobe Universities
1-1 Rokkodai, Nada, Kobe, Hyogo 657-8501, Japan
[1] {yanbenjp, yhplisajp}@gmail.com
[2] masa-n@cs.kobe-u.ac.jp

Abstract

The recent ubiquitous/pervasive technologies allow general household appliances to be connected within the network at home which is named home network system (HNS, for short). The great advantage of HNS is that it provides more value-added and powerful services by integrating multiple appliances and various sensors. Especially, sensor applications in HNS become much more important technology to build a high-level HNS service. In our earlier study, we have proposed the sensor service framework (SSF, for short) in the home network system for developing context-aware service, which wraps various sensor devices by web services to achieve easy development of context-aware service. In the SSF, a context was defined by a condition over a single sensor, or multiple sensors that derived by logical or arithmetic operations. However, the contexts were limited to the ones that can be defined by current values of the sensors, and can not describe the timing constraint relation in context on using multiple sensor web servers such as "after opening the door for 2 seconds, passed the hall" or "setting on the sofa", and hindered us from creating high-level timing constraints context.

In this paper, we propose a method for deriving the timing constraints context bases on the extended study of SSF. We first divide timing constraint in the context into two types: the sequential timing constraint and the continuous timing constraint. By using two types of timing constraint, the high-level context can be defined as conditions by using multiple sensors. After this, we also present a timer service to implement the timing constraints context within the SSF. We finally demonstrate how the high-level contexts with the timing constraints are registered and detected in a real home network system.

***Keywords:** home network system (HNS), context-aware services, sensor service framework (SSF), timing constraints context, sequential timing constraint, continuous timing constraint*

1. Introduction

The ubiquitous/pervasive technologies allow general household appliances to be connected within the network at home which is named Home Network System (HNS, for short). The great advantage of HNS is that it provides more value-added and powerful HNS services by integrating multiple appliances and various sensors. Recently, to provide high-level HNS service, the study about the development of context-aware application in HNS is becoming more and more notable [1][6]. The context refers to an information collection which contains the status of user, machine, surrounding environment, and various entities in HNS. It is always defined as a conditional expression by using the value over a single or multiple sensors. The context-aware application means executable software which can be implemented when a context is established.

In our earlier study [7], we proposed a Sensor Service Framework (SSF, for short) based on the idea of service-oriented architecture (SOA, for short) [8]. The SSF can make a developer use the sensor device (such as temperature sensor, illumination sensor etc.) as a standard web service by using the standard interfaces in HNS, such as getting value from sensor, registering context condition etc.. Since the SSF does not depend on any device or platform, it allowed loose coupling between applications, appliances and sensors, and the developer can integrate arbitrary appliances and sensors easily by using web service for developing the context aware application in HNS.

After this, we also propose the Sensor Mush up Platform (SMuP, for short) based on the SSF. The SMuP is used to derive more complex context which combines multiple sensor web services, such as the context [Temperature is 28 degrees or more and humidity is 40% or less] which integrates temperature sensor and moisture sensor, and which can be presumed as an advanced context.

However, the context presumed by SSF/SMuP is defined as a condition expression over a single or multiple sensors by using logical or arithmetic operations, and is limited to using the current values of the sensors. Thus, we can not describe the timing constraints contexts such as [ex.: after opening the door for 2 seconds, passed the hall], [ex.: setting on the sofa].

In this paper, the goal is to propose a method to define and presume the high-level timing constraints context as an extension of our earlier study of SSF/SMuP. We first define the timing constraint into 2 types, the sequential timing constraints and the continuous timing constraints. The sequential timing constraints context is a sequential time limitation between two contexts and means a relation like this: one context is detected within few seconds after another context is detected. The continuous timing constraints context is a continuous time limitation and means a context is continuously established during a period of time. The high-level context can be defined by combining such timing constraints between multiple contexts. In order to measure the time for the above timing constrains context, we also proposed the timer service as a web service to implement them. To evaluate the effectiveness of the proposed method, we have implemented two contexts (the Entering context and the Sleeping context) in testing environment CS27-HNS. As a result, the above timing constraints context can be registered and detected in CS27-HNS by using our proposed method.

2. Preliminaries

2.1. Home Network System (HNS)

A HNS consists of one or more networked appliances connected to a LAN at home. In general, each appliance has a set of application program interfaces (APIs), by which the users or external software agents can control the appliance via the network. A HNS typically has a home server, which manages all the appliances in the HNS. Services and applications are installed on the home server. A HNS service provides a sophisticated and value-added service by using multiple appliances together. The HNS service is implemented as a software application that invokes the APIs of the appliances. The appliances and services are deployed in a home, which is characterized by environmental attributes (e.g., temperature, humidity, brightness, current, sound, space) [11-[12].

In our library, based on service-oriented architecture (SOA) [4], we are developing a HNS texting environment [CS27-HNS] which can use various appliances as web service [10]. The appliances-dependent control method and communication protocol are wrapped by web service, and all appliance in CS27-HNS can be used as a web service of SOAP or REST formality. For example, we can set TV into 6ch by accessing URL: [http://cs27-hns/TVService/setChannel?channel=6].

2.2. Sensor Service Framework (SSF)

The sensor service framework (SSF) [2] was developed as an application framework for deploying sensor devices as web service in CS27-HNS easily. The web service wraps sensor-specific control logic into standard API. Each sensor has the measurable property. For example, the temperature sensor has a property *temperature (℃)*, the light sensor has a property *brightness (lux)*, and the value of these properties can be acquired from *getValue ()* method in CS27-HNS (Figure 1).

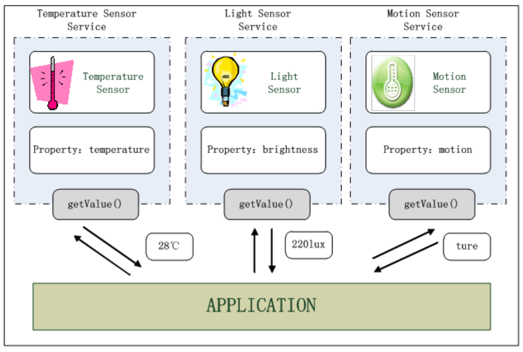

Figure 1. Standard API in SSF

Moreover, by implementing periodic observation of the change of sensor service's properties, the context can be detected based on the registered conditional expression (Figure2). For example, we defined a context named "hot" and registered it as context condition expression: "temperature > 27" (join context name and condition together, we write the above context as [hot: temperature >27]). The temperature sensor keeps observing "temperature" and detects context "hot" when the value of the "temperature" became higher than 27℃. The registration of context condition is implemented as "register ()", and the context can be called from any web service with using "subscribe ()" method. By using this web service, we can build context-aware service easily just by appointing am appliance of HNS. For example, the HNS service of "when the room is hot, the air-conditioner will turn on" can be implemented as [subscribe (hot, http://cs27-hns/AirConditionerService/on)]

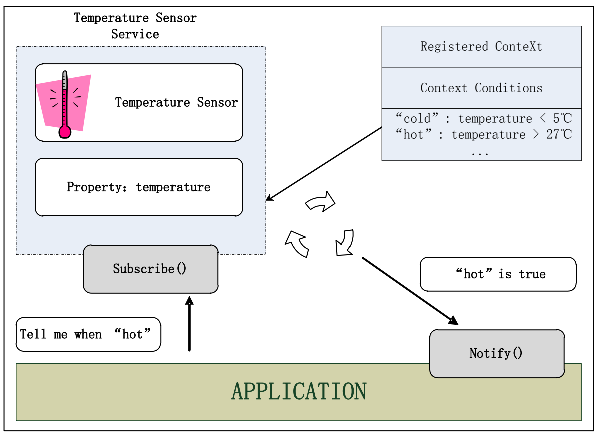

Figure 2. Registered Conditional Expression and Context Presumed

2.3. Sensor Mash-up Platform (SMuP)

In SSF, we note that the sensor service is always limited to the use of sensor. In order to build complex service by using various sensors, we developed the Sensor Mash-up platform (SMuP). The SMuP can create a virtual sensor service dynamically as an arithmetical or a logical operation on combining the property of existing sensor service or the condition of existing context. For example, take the average value of brightness by using multiple light sensors, combining temperature sensor and moisture sensor together to create a comfortable sensor. Based on this idea, the SMuP can build and detect more complex context easily, which can not be detected by using one sensor in SSF.

The virtual sensor can be created by *createSensor ()* method, and the *addProperty()* method can create new property for virtual sensor by combining exiting sensors. Because the virtual sensor can be implemented as a sensor service, it can be implemented by using *register ()* and *subscribe ()* method, which is the same as sensor service of SSF.

3. Research Goal and Approach

In our earlier study (SSF/SMuP), the condition of context was limited to the use of the current value of a sensor; it does not support to define the timing constraints context on using multiple sensors.

For example, we can not define and detect the flow contexts by using SSF/SMuP:

(P1: came home): [after opening the door for 2 seconds, passed the hall]

(P2: have a rest): [setting on the sofa]

(P3: begin to watch TV during a break): [setting on the sofa and begin to watch TV]

Under this circumstance, our goal is to propose a method to derive the timing constraints context as an extension of the earlier study of SSF/SMuP. To achieve this goal, we first analyze the timing constraints analysis and classify the timing constraints context into two types:

[**Type1**]: Sequential Timing Constraints Context

[**Type2**]: Continuous Timing Constraints Context

After this, we also propose a timer service which can measure the time of the timing constrains context. At last, we implement cases study in our testing environment CS27-HNS to prove the proposed method is effective.

4. Proposed Method

4.1. Key Idea

The key idea of this paper is to define timing constrains into two types. One is "the sequential timing constraint"; another is "continuous timing constraint". The timing constraint means a limited relation of times for one or multiple contexts must be satisfied. For example, the detection conditions of context described in chapter 3 can be defined as the flowing timing constrains:

- P1 (came home): [The motion sensor of entrance reacts within 5 seconds after the door sensor reacts.]

- P2 (have a rest): [The motion sensors of a sofa continue reacting for 15 seconds.]
- P3 (begin to watch TV during a break): [The TV is set to ON within 10 seconds after the motion sensors of a sofa react for 15 seconds.]

4.1.1. Sequential Timing Constraints： The sequential timing constraints is a time limited relation that a context ***Cx*** is detected within ***n*** seconds after another context *Cy* is detected. *T* is a new context with join *Cx* and *Cy*.

[*T* : *#n* [*Cx* , *Cy*]]

For example, a context [*C1: opening door*] detected by the door sensor of entrance can be defined as [*EntranceOpen: DoorSensor.isOpen == true*], another context [*C2: people in entrance*] detected by the motion sensor of entrance can be defined as [*HumanDetect: MotionSensor1.motion==true*], then the detected condition *P1* of the new context [*T1: coming home*] in 4.1 can be defined as

[*ComingHome* : *#5* [*EntranceOpen, HumanDetect*]]

On the other hand, reverse the turn of *C1* and *C2* above, another detected condition of a new context [*T2 : Leaving home*] can be defined as

[*LeavingHome* : *#5* [*HumanDetect, EntranceOpen*]]

Moreover, the context including more than three can also be expanded in to new timing constraints context. For example, join a new context [*C3: people in washroom*] which is detected by the motion sensor of the washroom into context *T1*, the detected condition of *C3* can be defined as [*WashRoom: MotionSensor2.motion==true*], then a new context [*T3: washing hands*] can be created and defined as

[*WashHands*: *#60* [*ComingHome, WashRoom*]]

4.1.2. Continuous Timing Constraints： The continuous timing constraints is a time limitation that a context ***Cx*** continues being established during period of time ***n***. ***T*** is a new context.

[*T* : *@n*[*Cx*]]

For example, a context [*C4: somebody is setting on sofa*] detected by the setting sensor of sofa can be defined as [*HumanSitting: SittingSensor.isSitting == true*], then the detected condition *P2* of the new context [*T4: have a rest*] in 4.1 can be defined as

[*Rest*: *@15* [*HumanSitting*]]

Moreover, the logic operation between multiple contexts can be used in the above formula too. For example, a context [*C5: people is not in living room*] detected by the sensor of living room can be defined as [*NotHumanDetect: MotionSensor3.motion == false*], another context [*C6: air-conditioner is working*] is defined as [*ACWorking: AC.power==true*], then a new context [*T5: WastePower*] for power saving can be defined as

[*WastePower*: *#600* [*NotHumanDetect && ACWorking*]]

4.1.3. Combination of Timing Constraints Context: Our proposal is also applicable to the combination of sequential and continuous timing constraints contexts to create for creating much more complex application. For example, for a context [*C7: TV on*] is defined as [*TVon : TV.power==true*], by joining the context [*T4:have a rest*] that was defined in 4.3, a new context [*T6: have a rest and watching TV*] can be defined as

[*WachingTV* : *#10* [*@15*[*HumanSetting*] , *TVon*]]

4.2. Registration and Detection of Timing Constrains Context with Time Service

In order to implement the timing continuous context in HNS, we also proposed a time service for time measurement which consists of 4 methods and 3 states.(Table 1)

Table 1. Main Components of Timer Service

Method	createTimer(*n*)	Create a timer object for measuring time *n*.
	start(),	Start to measure time *n*.
	stop()	Stop to measure time *n*.
	isActive()	Get the status of timer object.
state	Waiting	The timer object does not start to measure time *n*.
	Working	The timer object is measuring time *n*.
	Expired	The timer object is time-out.

4.3. Registration and Detection Steps for Timing Constrains Context

In this section, we will show how to register and detect the timing constrains context by using sensor service and timer object of SSF/SMuP. It includes 4 steps as below (Figure 3).

- **STEP1**: Create a virtual sensor ***S*** for detecting timing constrains context ***T***;
- **STEP2**: Create a timer service for measuring time ***n*** by using time service;
- **STEP3**: Start to measures time ***n***;
- **STEP4**: Define the detecting condition and add it to ***S*** as a property.

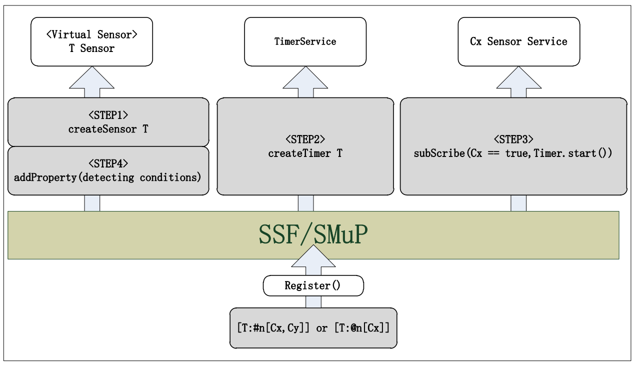

Figure 3. The Registration and Detection Steps of Timing Constrains Context

For example, for the sequential timing constraints context [*T: #n* [*Cx* , *Cy*]], we have a registration requirement of context [*T1:coming home*] which is defined as

[*ComingHome:* #5 [*EntranceOpen*, *HumanDetect*]]

Following the steps above, the registration steps of this context are as below:

- STEP1: Creating a virtual sensor *ComingHomeSensor* by using *createSensor()* method;
- STEP2: Creating a timer service (*CHTimer*) for measuring 5 seconds by using *createTimer(5)* method of time service;

- ■ STEP3: Implementing *subscribe(EntranceOpen, CHTimer.start)* method of the *DoorSensorService* which is registered by context *EntranceOpen*, and connect *CHTimer.start()* with *EntranceOpen*;
- ■ STEP4: Adding a new property (*comingHome: HumanDetect ==true && CHTimer.isActive==working*) to the virtual sensor *ComingHome sensor*.

Then property *ComingHome* of *ComingHomeSensor* become to true only when *HumanDetect* is established within 5 seconds after *EntranceOpen* is established. The property *ComingHome* can be acquired through *getValue()* method of *ComingHomeSensor* which was made at the time of context *T1* registration (Figure4:T1 Coming home) .

For the continuous timing constraints context [*T: @n*[*Cx*]], we suppose that we have a registration requirement of context [*T4:have a rest*] which is defined as

[*Rest: @15* [*HumanSitting*]]

and this context can be registered as below:

- ■ STEP1: Creating a virtual sensor *RestSensor* by using *createSensor()* method;
- ■ STEP2: Creating a timer (*RTimer*) for measuring 15 seconds by using *createTimer(15)* method of time service;
- ■ STEP3: Implementing *subscribe(HumanSitting, CHTimer.start)* method of the *SittingSensorService* that was registered by context *HumanSitting*, and connect *CHTimer.start()* with *HumanSitting*;
- ■ STEP4: Adding a new property (*Rest: HumanSettingt ==true && RTimer.isActive==Expired*) to the virtual sensor *RestSensor*.

Then property *Rest* of *RestSensor* becomes true only when *HumanDetect* continues establishing for 15 seconds. The property *Rest* can be acquired through *getValue()* method of *RestSensor* which was made at the time of context *T4* registration (Figure4:T4 have a rest).

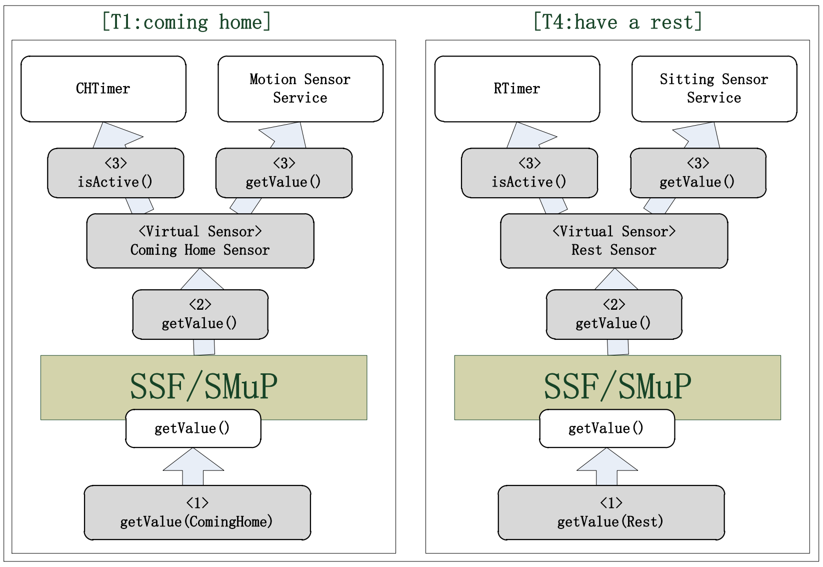

Figure 4. The Example of Registration and Detection Steps for [T1: coming home] and [T4: have a rest]

5. Case Study

5.1. Implementation of Time Service

To derive high-level timing constraints, we implemented the timer service as a web service and deployed it in our HNS testing environment CS27-HNS. The developing environment and the technique we used as follows:

■ Development Tools:	Eclipse3.5.2
■ Development Language:	Java(JRE1.5.0_18)
■ Web Server:	Apache Tomcat 5.5
■ Web Service Engineer:	Apache Axis 2.1.3

5.2. Case Study

As a case study, this section shows two timing constraints cases which were implemented in our HNS testing environment CS27-HNS. One is a sequential timing constrains context [***T:Entering***], another is a continuous timing constraints context [***T:Sleeping***].

CS27-HNS is a testing home network system environment in Kobe University. For this testing case, we use 2 motion sensors (Motion Sensor 1111) and 3 force sensors (Force sensor 1106) of Phidgets Company to detected the above two contexts. Each sensor can be used as a web sensor service based on SSF/SMuP.

5.2.1. Implementation of Context [T:Entering]: Two motion sensor devices are deployed on entrance and living room of CS27-HNS. Each sensor can be accessed as a web sensor service: ***MotionSensorService1*** (entrance) and ***MotionSensorService2*** (living room), and each service has a ***motion*** property to judge that there is a person or not. The context ***HumanDectect1*** and ***HumanDectect2*** are registered as

[***HumanDectect1: MotionSensorService1.motion==true***]
[***HumanDectect2: MotionSensorService2.motion==true***].

Then the detected condition of context [***T:Entering***] is defined as [***Entering*: **#5[***HumanDetect1***, ***HumanDetect2***]], it means that "the motion sensor of living room reacts within 5 seconds after the motion sensor of entrance reacts". Following the steps that we proposed in section 4-3, the context [***T:Entering***] can be registered as below.

- **STEP1**: Create a virtual sensor
http://cs27-hns/SMuP/createSensor?name=EnterHomeSensor
- **STEP2**: Create a timer
http://cs27-hns/TimerService/createTimer?name=EHTimer&time=5
- **STEP3**: Connect MotionSensor1 and timer
http://cs27-hns/MotionSensorService1/subscribe?context=Human Detect1&notify =http://cs27-hns/ETimer/start
- **STEP4**: Register HumanDetect2 and timer to context T as property
http://cs27-hns/SMuP/EnterLeaveSensor/addProperty?name=EnterHome &property=HumanDetect2==true && EHTimer==Working

Then the context [***T:Entering***] can be detected by calling "http://cs27-hns/SMuP/EnterLeaveSensor/getValue?property=Enter".

5.2.2. Implementation of Context [T:Sleeping]: The context [***T:Sleeping***] is implemented by using 3 force sensors which is setting on left, middle and right side of sofa in CS27-HNS. Each force sensor has a property (SittingL, SittingM and SittingR) which can be accessed as a web service for judging a person is setting on there or not. Because the detected condition of [***T:Sleeping***] is that the force sensors of sofa continue reacting for 60 seconds, so it can be defined as [***Sleeping:@60[SittingL&&SittingM&&SittingR]***].

Following the steps that we proposed in section 4-3,, the context [***T:Sleeping***] can be registered as below steps.

- **STEP1**: Create a virtual sensor
  http://cs27-hns/SMuP/createSensor?name=SleepingSensor
- **STEP2**: Create a timer
  http://cs27-hns/TimerService/createTimer?name=STimer&time=60
- **STEP3**: Connect Force Sensors and timer
  http://cs27-hns/SMuP/createSensor?name=AllForceSensor
  http://cs27-hns/SMuP/AllForceSensor/addProperty?name=AllPress& property=SittingL==true && SittingC==true && SittingR==true
  http://cs27-hns/SMuP/AllForceSensor/subscribe?context=AllPress& notify=http://cs27-hns/STimer/start
- **STEP4**:Register AllPress property and timer to virtual sensor
  http://cs27-hns/SMuP/SleepingSensor/addProperty?name=Sleeping& property=AllPress==true && STimer==Expired

Then the context [***T:Entering***] can be detected by calling "http://cs27-hns/SMuP/SleepingSensor/getValue?property=Sleeping".

6. Conclusions and Future Work

6.1. Conclusions

In this paper, we have presented a method for detecting the high-level timing constraints context based on the sensor service in HNS. In this proposed method, the timing constraints context was defined into 2 types: the sequential timing constraints context and the continuous timing constraints context. The sequential timing constraints context is a sequential time limitation between two contexts and means a relation such as one context is detected within few seconds after another context is detected. The continuous timing constraints context is a continuous time limitation that means a context continuously being established during a period of time.

In order to measure the time for the above timing constrains context, we also proposed the timer service as a web service to implement them. To evaluate the effectiveness of the proposed method, we have implemented two contexts (the Entering context and the Sleeping context) in testing environment CS27-HNS. As a result, the above timing constraints context can be registered and detected in CS27-HNS by using our proposed method.

6.2. Future Work

Depending on the proposed method above, although the more complicated context can be detected, the creation work of context conditions itself is complicated. The current limitation is that for creating high-level context with timing constraints, the creator needs to know the details about the pre-existing context of HNS, and also needs to have the ability to analyze and implement complex logic to detect the high-level timing constraints context.

Thus, in the future, we plan to develop a framework to create the high-level context with timing constraints. This framework should be a web service for collectively managing the information of a pre-existing context of HNS. When the developer uses a pre-context to create high-level context, it isn't necessary to call each sensor service as before, the goal is to make the development of the context aware application with a new timing constraints context easier than before.

Acknowledgements

This research was partially supported by the Scientific Research Start Funds of LuoYang Institute of Science and Technology (No.14308051); the Japan Ministry of Education, Science, Sports, and Culture [Grant-in-Aid for Scientific Research (B) (No.26280115, No.15H02701), Young Scientists (B) (No.26730155)].

References

[1] B. N. Schilit, N. Adams and R. Want, “Context-Aware Computing Applications”, Proceedings of the 1st IEEE Workshop on Mobile Computing Systems and Applications (WMCSA), **(1994)** December 8-9, Washington, DC, USA.

[2] A. K. Dey and G. D. Abowd, “Towards a Better Understanding of context and context-awareness”, Proceeding of the 1st International Symposium on Handheld and Ubiquitous Computing (HUC), **(1999)** September 27-29, Karlsruhe, Germany.

[3] H. Sakamoto, H. Igaki and M. Nakamura, “A Sensor Service Framework for Context-aware Applications”, Technical Report Of The Institute Of Electronics, Information And Communication Engineers, vol. 108, no. 458, **(2009)** March, pp. 381-386.

[4] T. Erl, “Service-Oriented Architecture, Concepts, Technology and Design”, Prentice Hall **(2008)**.

[5] H. Sakamoto, H. Igaki and M. Nakamura, “SMuP, A Service-oriented Platform for Sensor Service Mashups”, Winter workshop 2010 in Kurasiki, vol. 2010, no. 3, **(2010)** January, pp. 73-74.

[6] A. Tanaka, M. Nakamura, H. Igaki and K. –I. Matsumoto, “Adapting Conventional Home Appliances to Home Network Systems Using Web Services”, Technical Report Of The Institute Of Electronics, Information And Communication Engineers, vol. 105, no. 628, **(2006)** March, pp. 67-72.

[7] M. Fukuda, H. Seto, H. Sakamoto, H. Igaki and M. Nakamura, “A Looking Back Service for Power Consumption Logs in Home Network System”, Technical Report Of The Institute Of Electronics, Information And Communication Engineers, vol.109, no. 272, **(2009)** November, pp. 29-34.

[8] M. Nakamura, H. Igaki, H. Tamada and K. –I. Matsumoto, “Implementing Integrated Services of Networked Home Appliances Using Service-oriented Architecture”, the Journal of Information Processing Society of Japan, vol. 46, no. 2, **(2015)** February, pp. 314-326.

[9] “Phidgets Inc. Unique and Easy to Use USB Interface”,http://www.phidgets.com/.

[10] M. Syuhei, S. Hideharu, S. Hiroyuki, H. Igaki and M. Nakamura, “Sensor search with spatial information and support by showing similar parameter for building sensor context”, technical report of the institute of electronics, information and communication engineers, ,vol. 109, no. 327, **(2009)** December, pp. 59-64.

[11] B. Yan, M. Nakamura, L. du-Bousquet, and K. -I Matsumoto, “Validating Safety for Integrated Services of Home Network System Using JML”, Journal of Information Processing (JIP), vol.49, no. 6, **(2008)** June, pp. 1751-1762.

[12] B. Yan, M. Nakamura and K. -I Matsumoto, “Deriving Safety Properties for Home Network System Based on Goal-Oriented Hazard Analysis Model”, International Journal of Smart Home, vol.3, no. 1, **(2009)** January, pp. 67-80.

[13] B. Yan, M. Nakamura, L. D. Bousquet and K. -I Matsumoto, “Improving Reusability of Hazard Analysis Model with Hazard Template for Deriving Safety Properties of Home Network System”, International Journal of Smart Home, vol.3, no. 2, **(2009)** April, pp.71-88.

[14] L. D. Bousquet, M. Nakamura, B. Yan and H. Igaki, “Using Formal Methods to Increase Confidence in a Home Network System Implementation”, a Case Study, Innovations in Systems and Software Engineering (ISSE Journal), vol. 5, no. 3, **(2009)** September, pp. 181-196.

Authors

Ben Yan, received the B.E. degree in Henan University of Science and Technology, China, in 1999, M.E. degree in Department of Information Science Okayama University of Science, Japan, in 2006, and Ph.D. degree in the Graduate School of Information Science at Nara Institute of Science and Technology, Japan, in 2008. From 2009 to 2014, he worked for Panasonic Group, SANYO Information Technology Solutions Co., Ltd, and Osaka, Japan. He is currently a professor in the Department of Computer and Information Engineering at Luoyang Institute of Science and Technology (LIT). His main research interests include the service-oriented architecture, the V&V of home network systems, and requirements engineering for safety critical systems.

HuaPing Yao, received the B.E. degree in Henan University of Science and Technology, China, in 1999, M.E. degree in Department of Information Science Okayama University of Science, Japan, in 2006. From 2006 to 2014, he worked for CSI and Trend Creates Co., Ltd, Osaka, Japan. She is currently a lecturer in the Department of Computer and Information Engineering at Luoyang Institute of Science and Technology (LIT). Her main research interests include the e-learning, software engineering and home network system.

Masahide Nakamura, received the B.E., M.E., and Ph.D. degrees in Information and Computer Sciences from Osaka University, Japan, in 1994, 1996, 1999, respectively. From 1999 to 2000, he has been a post-doctoral fellow in SITE at University of Ottawa, Canada. He joined Cyber media Center at Osaka University from 2000 to 2002. From 2002 to 2007, he worked for the Graduate School of Information Science at Nara Institute of Science and Technology, Japan. He is currently an associate professor in the Graduate School of System Informatics at Kobe University. His research interests include the service /cloud computing, smart home, smart city, and life log. He is a member of the IEEE, IEICE and IPSJ.

Shinsuke Matsumoto, received the B.E. degree in computer science from Kyoto Sangyo University in 2006. He received M.E. and Ph.D. degrees in information science from Nara Institute of Science and Technology in 2008 and 2010, respectively. He is currently an assistant professor in the Graduate School of System Informatics at Kobe University. His research interests include software engineering, mining software repository and cloud computing.