Computer Networks 57 (2013) 2442–2453

Contents lists available at SciVerse ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Considering impacts and requirements for better understanding of environment interactions in home network services

Masahide Nakamura *, Kousuke Ikegami, Shinsuke Matsumoto

Graduate School of System Informatics, Kobe University, Japan

ARTICLE INFO

Article history:
Received 24 September 2012
Received in revised form 19 February 2013
Accepted 22 February 2013
Available online 22 April 2013

Keywords:
Home network system
Integrated services
Environment feature interactions
Finite state machines
Environment impact
Environment requirement

ABSTRACT

A home network system (HNS) coordinates various networked home appliances to achieve value-added services. If multiple services are executed at the same time, functional conflicts between the home appliances may occur. These are known as feature interactions (FIs) in the HNS. We have previously defined two kinds of FIs: *appliance interactions* and *environment interactions*. Environment interaction refers to an indirect conflict of different appliances in the home environment, which is generally more difficult to capture than appliance interaction. Due to a lack of an amount of environmental impacts and requirements to be satisfied, the previous definition missed some obvious environment interactions, or mis-detected many acceptable cases.

In this paper we try to extend the previous formalization by introducing two new concepts. First we propose an *environment impact model*, which strictly defines how each appliance operation contributes to the environment properties. Second, we introduce an *environment requirement* to define the expected environment state achieved by each service. We then re-formalize the environment interaction by a condition such that the accumulated impacts violate the requirement of either of the services. A case study with five practical services successfully detects the interactions that could not be characterized by the previous definition.

1. Introduction

A *home network system (HNS)* (also called a smart home) is an emerging ubiquitous application that intends to provide smart and convenient home services [1,2]. The HNS consists of networked household appliances and environmental sensors. They include TVs, lights, air-conditioners, curtains and thermometers, which are all connected to the home network. The features of each appliance can be accessed via an API. By executing the APIs of multiple appliances using a certain service logic, the HNS achieves *integrated services*. Here we introduce some examples.

Coming Home Service (CH) welcomes a user coming home. When a user comes home the service turns on the lights in an entrance hall and a living room, activates an aroma diffuser with a relaxation fragrance, and turns on an air-conditioner.
TV Theater Service (TVT) sets up a living room for watching TV in a theater-like atmosphere. When the service is activated, a curtain is closed, lights are turned off, and the TV is turned on.
BGM Service (BGM) provides back ground music using a music player in a living room.
Air Cleaning Service (AC) cleans and deodorizes air in the room using an air-cleaner. It also opens a window to help air-cleaning.

As it can be easily imagined these services cause *feature interactions* (*FIs*) [3,4] when they are executed

* Corresponding author. Tel.: +81 78 803 6398; fax: +81 78 803 6295.
E-mail address: masa-n@cs.kobe-u.ac.jp (M. Nakamura).

1389-1286/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.comnet.2013.02.024

simultaneously in the same environment. Instances of feature interactions among the above integrated services are listed as follows.

FI-1 (CH vs TVT): When the CH and the TVT are executed at the same time, a conflict occurs concerning the light in the living room. Specifically, the CH turns on the light while the TVT turns off the same light.

FI-2 (CH vs TVT): Another interaction can be observed for the same combination of services. The CH turns on the light in the entrance hall. The light indirectly makes the living room brighter, which may ruin the theater-like atmosphere of the TVT.

FI-3 (TVT vs BGM): This interaction occurs in the audio domain. While the user is watching a movie with the TVT, if someone activates the BGM the music disrupts the sound of the movie.

FI-4 (CH vs AC): Two services have opposite requirements on the fragrance in the room. The CH selects the aroma diffuser for the purpose of relaxation, while the AC uses the air-cleaner to remove any kind of odor.

In our previous work [5] we defined two kinds of feature interactions in the HNS: *appliance interactions* and *environment interactions*. In our definition appliance interaction occurs when two services request incompatible operations on the same appliance. For instance, FI-1 described above is an appliance interaction, since incompatible operations `on()` and `off()` are requested of the same living-room light. The mechanism of the appliance interactions is simple and intuitive. Therefore we were able to formalize them by reasonable conditions. Based on this we developed offline and online detection systems for our practical HNS [6,7].

On the other hand, environment interaction occurs when two operations of different appliances indirectly interfere via environment properties (e.g. temperature, brightness, etc.). FI-2, FI-3 and FI-4 above are all environment interactions. In FI-2, the living-room light and the entrance light conflict over the brightness in the living room. As for FI-3, the music player disturbs the TV with respect to the sound of the movie. In FI-4, the effect of the aroma diffuser is nullified by the air-cleaner.

Compared to appliance interactions, it is generally more difficult to formalize environment interactions. In our original definition [5] environment interactions were simply characterized by conflicting read/write operations regarding a certain environment property. Matsuo et al. [8] extended our definition by introducing the direction of effect to the write operation. Kolberg et al. [4] and Wilson et al. [9] used resource-locking mechanisms to express environment interactions. Metzger et al. [10] modeled the dependency between the control and the environment with a goal-oriented framework.

However, we found that these previous definitions lack sufficient detail to soundly capture all environment interactions. First, they did not consider the degree (or amount) of environmental impact. Our original definition only dealt with the dependency between an operation and the environment only but did not consider *how much impact* is posed by the dependency. Thus we could not distinguish, for example, the case of FI-2, where the effect of the entrance light on the living room is very small. Second, the previous definitions did not consider the explicit requirement expected by each service on the environment. Therefore, there were no criteria to determine whether the environment interactions detected are *acceptable or not*. For instance, FI-4 may not be a feature interaction if the aroma fragrance is not a primary requirement of CH.

The goal of this paper is to offer a new definition of environment interactions in the HNS that overcomes the above limitations. To cope with the first problem we propose an *environment impact model*, which strictly models the impact of each appliance operation on the environment's properties. We then define an *environment requirement* to address the second problem, which specifies a primary requirement to be achieved by each service. Finally, we provide the new definition of environment interactions. Intuitively, integrated services S_1 and S_2 cause an environment interaction should the overall environmental impacts (computed by the impact model) violate a certain environment requirement of S_1 or S_2.

To evaluate the proposed method we conduct a case study with five practical services. The results show that the proposed method successfully detects those environment interactions that cannot be characterized by the previous definition.

2. Feature interactions in home network system

We first review feature interactions in the HNS based on the previous studies.

2.1. Previous HNS model

In our previous work [5], we proposed an object-oriented model to formalize feature interactions in the HNS. In the model, every appliance in the HNS is represented by an *object* with *properties* and *methods*. Each property is a variable holding a certain value ranging over a type, which characterizes the internal state of the appliance. Each method m represents the API of the appliance, defined by the following elements.

Pre(m): a *pre-condition* that should be satisfied before executing m. It is defined with the appliance properties.

Post(m): a *post-condition* that should hold after executing m. It is defined with the appliance properties.

EnvRead(m): a set of environment properties that are referenced by m.

EnvWrite(m): a set of environment properties that are overwritten by m.

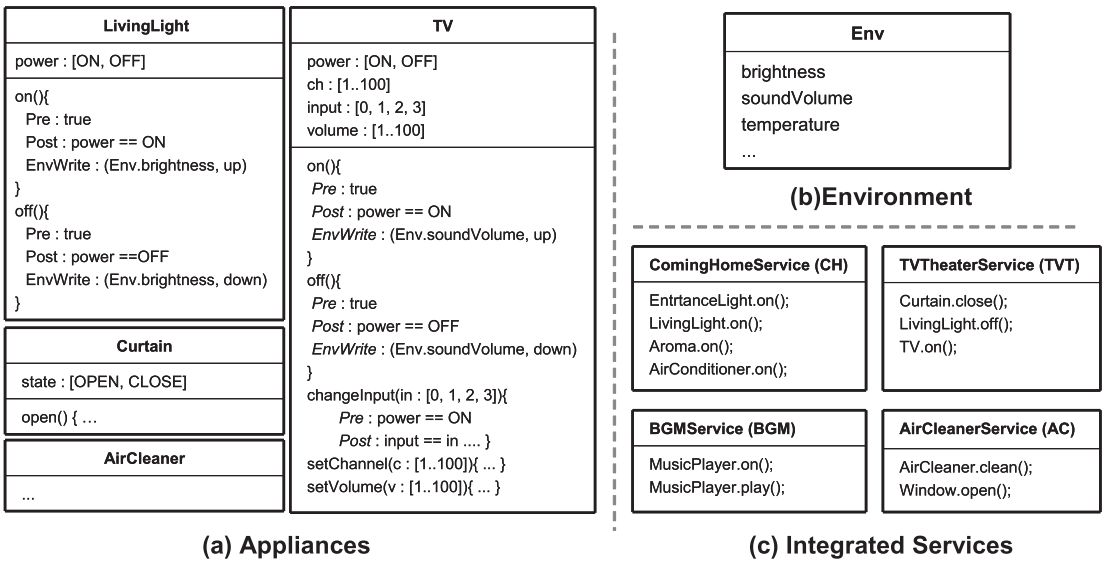


Fig. 1. Previous HNS model.

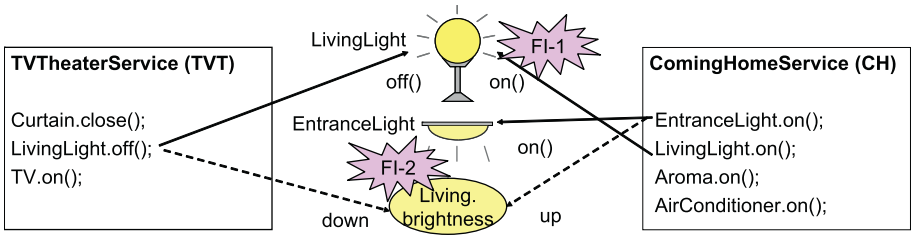


Fig. 2. Example of appliance interaction (FI-1) and environment interaction (FI-2).

The pre/post-conditions characterize the functional aspect of the API within the appliance. EnvRead/EnvWrite briefly represents the effects from/to the HNS environment, respectively. Basically, a method for sensing the environment has EnvRead elements, while a method for affecting the environment has EnvWrite elements. Matsuo et al. [8] elaborated our original definition by introducing *direction* in EnvWrite, in order to explicitly represent the direction of the effect. The environment is a global object consisting of environment properties, which are referenced by all objects and services. An integrated service is defined by a sequence of appliance methods.

Fig. 1 shows an example of the HNS model. Fig. 1a represents the appliance model, consisting of LivingLight, TV, Curtain and AirCleaner. Let us take TV for example. This TV has four properties: `power`, `ch`, `input` and `volume`. `TV.power` can take a value of either `ON` or `OFF`. `TV.sound` is an integer ranging from 1 to 100. `TV.on()` has a pre-condition `true`, indicating it can be executed at any time. If `TV.on()` is executed, `TV.power` becomes ON as specified in the post-condition. The EnvWrite `(Env.soundVolume, up)` indicates that `TV.on()` increases the sound volume of the environment.

Fig. 1b shows an example of the environment object, consisting of some typical environment properties: `brightness`, `soundVolume`, `temperature`, etc. Fig. 1c presents a model of the four integrated services, CH, TVT, BGM and AC, which are the same services introduced in Section 1. For example, the TVT model is defined by three appliance methods: `Curtain.close(); LivingLight.off(); TV.on();`. The sequence of the methods sets up the theater-like atmosphere for watching TV in the living room.

We assume that the HNS model is defined for every *isolated* room in the house. Unless explicitly specified, every environment property of a room is *independent* of those of another room (or outside the house). Also, we do not count the *side-effects* of an appliance method. For example, turning on an air-conditioner may make a noise, increasing the sound volume in the room. However, the effect on the sound volume is not considered, unless explicitly specified in the model.

2.2. Previous formalization of feature interactions

Using the HNS model we have defined the following two kinds of feature interaction in the context of the HNS. In the following, let S_1 and S_2 be any integrated services, and let m_1 (or m_2) be any appliance method executed in S_1 (or S_2, respectively).

Appliance Interaction: m_1 and m_2 are methods of the same appliance d, and are incompatible with each other. Specifically, we state that S_1 and S_2 cause appliance interaction if at least one of the following conditions is satisfied.

- $Post(m_1) \wedge Post(m_2) = \Phi$ (unsatisfiable) or
- $Post(m_1) \wedge Pre(m_2) = \Phi$ (unsatisfiable)

Environment Interaction: m_1 and m_2 are methods of different appliances d_1 and d_2, respectively. However, m_1 and m_2 indirectly conflict via a certain environment property *Env.e*. Specifically, we say that S_1 and S_2 cause environment interaction if an environment property *Env.e* exists such that:

- $(Env.e, up) \in EnvWrite(m_1) \wedge (Env.e, down) \in EnvWrite(m_2)$ or
- $(Env.e, up\ or\ down) \in EnvWrite(m_1) \wedge (Env.e) \in EnvRead(m_2)$

Note that the condition is asymmetric for S_1 and S_2, because interchanging m_1 and m_2 in the second condition of each definition cause a different situation[1]. So, we have to consider the direction of combinations. For example, for TVT and CH we have to check two combinations [TVT vs CH] and [CH vs TVT] to detect the feature interactions.

Fig. 2 explains the feature interactions based on the formalization. In the figure, FI-1 and FI-2 are respectively the appliance and environment interactions explained in Section 1. In the following sections we especially focus on environment interactions.

2.3. Limitations of previous definition of environment interaction

In our subsequent research we found some cases of environment interactions that could not be explained reasonably with the above definition. This was due to the following two limitations.

Limitation 1: The model for interacting with the environment is too coarse and abstract. For instance, let us take FI-3 (TVT vs BGM) based on the previous model. Both `TV.on()` of TVT and `MusicPlayer.on()` of BGM increase `Env.soundVolume`, and thus TVT and BGM do not cause the environment interaction. However this is against our intuition. As for FI-4 (CH vs AC) it is not obvious to specify the direction of fragrance change, since fragrance is not a numeric metric. After all, the previous model with environmental read/write operations only was too coarse-grained to capture actual feature interactions.

Limitation 2: The requirement of service is not considered at all. Since requirements of services (or users) are not considered at all, the previous definition regards many acceptable cases as feature interactions. For instance, consider FI-2 (CH vs TVT) again. If the effect of the entrance light on the living room is very small, this is an acceptable (or desirable) feature interaction. However, the previous definition had no way of distinguishing acceptable cases from harmful ones.

[1] Intuitively, the second condition of each definition states that the result of S_1 may disturbs the execution of S_2. Hence, for a given pair of services S_A and S_B, the definition distinguishes between two cases which of S_A or S_B disturbs another.

3. Characterizing environment impact in details

To cope with Limitation 1, in this section we propose a new model, called the *environment impact model*. The environment impact model describes the effect of each appliance method on the environment in fine detail.

3.1. Classification of environment property

To establish the environment impact model we first increase the detail level of the environment properties. We here classify every environment property into two types: *numerical* or *non-numerical*. A numerical property is an environment property that can be represented by a number. Typical numerical properties include `temperature`, `brightness`, `soundVolume`, `humidity`, etc. For a numerical property, arithmetic operations can be applied.

On the other hand, a non-numerical property is the one that is not numerical. Typical properties include `fragrance`, `TV content`, `music content`, etc. For non-numerical properties we cannot use arithmetic operations. Instead, we use set operations.

3.2. Environment impact

To precisely capture the environment interactions, it is essential to know, for every appliance method, *how much impact is given to which environment properties*. For instance, suppose that when `Light.on()` is executed the `Light` may increase by 200 lux. In this case we would like to associate the impact of `[Env.brightness += 200]` to `Light.on()`, where `+=` denotes a self-addition operator. Let us consider another example `AirCon.setTemperature(25)`, where the air conditioner tries to maintain the room temperature at 25°. In this case we see the impact `[Env.temperature := 25]`, where := represents an assignment operator.

As shown in the examples there are two types of impact: *cumulative* and *non-cumulative*. The cumulative impact adds (or subtracts) a certain value to (or from) the current value of an environment property. Let *e* be an environment property. We use the notation `[e += value]` (addition) or `[e -= value]` (subtraction) to represent a cumulative impact. On the other hand, the non-cumulative impact replaces the current value of an environment property with a new value. For this, we use the notation `[e := value]`.

The above impacts are valid when *e* is a numerical property. When *e* is non-numerical we use set operations. For a cumulative impact of a non-numerical property *e*, we use the notation `[e U={v1,v2,...}]` (set addition) or `[e \= {v1,v2,...}]` (set subtraction). For a non-cumulative impact, we use `[e := {v1,v2,...}]`, where := is an assignment of set. For example, take `fragrance` as an example of a non-numeric property then `Aroma.on()` in CH (see Fig. 1) has a cumulative impact `[Env.fragrance`

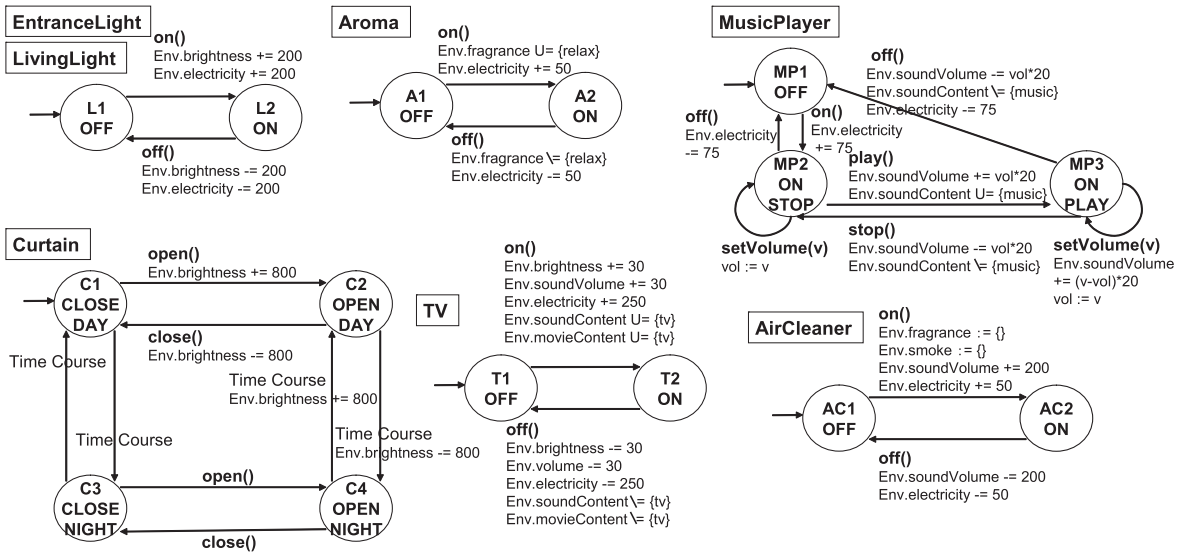


Fig. 3. Environment impact model.

`U={relax}]`, which adds the relaxation aroma to the environment. `AirCleaner.clean()` in AC has a non-cumulative impact `[Env.fragrance := {}]`, which freshens the room.

3.3. Environment impact model

In general, the environment impacts vary depending on the state of the appliances. Indeed, `Light.on()` does not increase the brightness when the light is already on. Therefore we introduce a finite state machine (FSM).

The proposed environment impact model consists of FSMs, each of which corresponds to an appliance. Let E be a set of environment properties, and I be a set of environment impacts. Also, let d be an appliance. Then, the environment impact model of d is defined by a FSM $I_d = (S_d, M_d, T_d, \Delta, s_0)$, where

- S_d: a set of states of d
- M_d: a set of methods of d
- $T_d \subseteq (S_d \times M_d \times S_d)$: a set of state transitions
- $\Delta(E \times T_d \to I)$: an environment impact function
- $s_0 \in S_d$: initial state.

The environment impact function Δ associates each state transition with an impact with respect to an environment property. We assume that all FSMs are in their initial states at the start. When a service executes a method m of an appliance d, a state transition t of I_d occurs.

Fig. 3 shows the environment impact model of the HNS appliances. For example, an FSM of TV has two states: OFF and ON. A transition `(T1, TV.on(), T2)` has the following five associated impacts:

- `[Env.brightness += 30] /* makes the room a bit brighter */`
- `[Env.soundVolume += 30] /* makes the room louder */`
- `[Env.electricity += 200] /* increases electricity consumption */`
- `[Env.soundContent U= {tv}] /* adds sound content to TV */`
- `[Env.movieContent U= {tv}] /* adds movie content to TV */`

We assume that the environment impacts are specified by the user of the proposed method. Each value may be determined by a specification of an appliance, a layout of the room, a floor plan of the house, or the outside environment. This paper does not go into detail of how to construct consistent models for individual houses.

3.4. Total environment impact caused by service

An integrated service generally contains several appliance methods. Hence as the service is executed, the several environment impacts associated with the methods are accumulated in the environment.

Thus, for an integrated service S and an environment property $e \in E$, we define the *total environment impact to be caused by S*, denoted by $TEI(e,S)$, as the sum of the degree of impacts given to e by S. Let $[t_1, t_2, \ldots, t_x]$ be a sequence of transitions caused by the execution of S. Then, $TEI(S,e)$ is defined by:

$$TEI(e,S) = \sum_{i=1}^{x} \Delta(e, t_i)$$

We also consider the total impact of multiple services. Let $S_1; S_2; \ldots; S_n$ be a sequence of all services currently executed in this order. Then the total environment impact on e is defined by:

$$TEI(e) = \sum_{j=1}^{n} TEI(e, S_j)$$

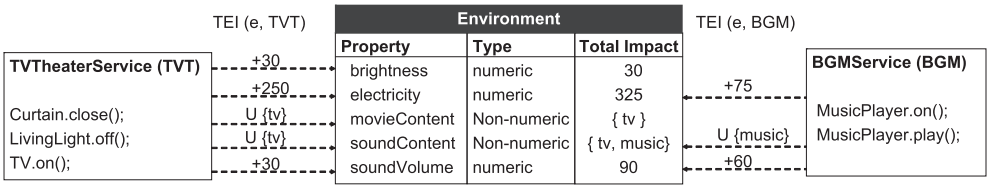


Fig. 4. Example of total environment impact.

TEI(*e*,*S*) characterizes how much environment impact is made on *e* by the execution of a single service *S*. *TEI*(*e*) represents how much total impact is currently made on *e* by all services. Here, a service *S* might be interested in how much impact is currently made by *other services excluding itself*. For this, we define the *exclusive environment impact XEI*(*e*,*S*) as:

$$XEI(e,S) = TEI(e) - TEI(e,S)$$

which is used to investigate the total impact caused by services other than *S*.

For example, let us consider the total environment impact caused by TVT. As shown in Fig. 1c, TVT executed three methods: `Curtain.close(); LivingLight.off(); TV.on();`. According to Fig. 3 we can calculate:

```
TEI(Env.brightness, TVT) = 0 + 0 + 30 = 30
TEI(Env.electlicity, TVT) = 0 + 0 + 250 = 250
TEI(Env.movieContent, TVT) = {} U {} U {tv} = {tv}
TEI(Env.soundContent, TVT) = {} U {} U {tv} = {tv}
TEI(Env.soundVolume, TVT) = 0 + 0 + 30 = 30
```

Similarly, we also calculate the total impact of the BGM, assuming that the sound volume (`vol`) of the music player is currently 3.

```
TEI(Env.electlicity, BGM) = 75 + 0
TEI(Env.soundContent, BGM) = {} U {music} =
  music
TEI(Env.soundVolume, BGM) = 0 + 3*20 = 60
```

Suppose that TVT and BGM are all the services currently executed. Then, we have

```
TEI(Env.brightness) = 30
TEI(Env.electlicity) = 250 + 75 = 325
TEI(Env.movieContent) = {tv}
TEI(Env.soundContent) = {tv} U {music} = {tv,
  music}
TEI(Env.soundVolume) = 30 + 60 = 90
```

Now we calculate the exclusive impacts of TVT, to see the impacts caused by services other than TVT.

```
XEI(Env.brightness, TVT) = 0
XEI(Env.electlicity, TVT) = 75
XEI(Env.movieContent, TVT) = {}
XEI(Env.soundContent, TVT) = {music}
XEI(Env.soundVolume, TVT) = 60
```

Fig. 4 schematically describes how much environment impacts are given by TVT and BGM. Compared to the previous model in Fig. 2 (FI-2), we can see that the proposed environment impact model can capture the effect of services on the environment in much more detail.

4. Introducing requirements of services

4.1. Environment requirement of service

As discussed in Limitation 2, the previous definition of the environment interactions did not consider a require-

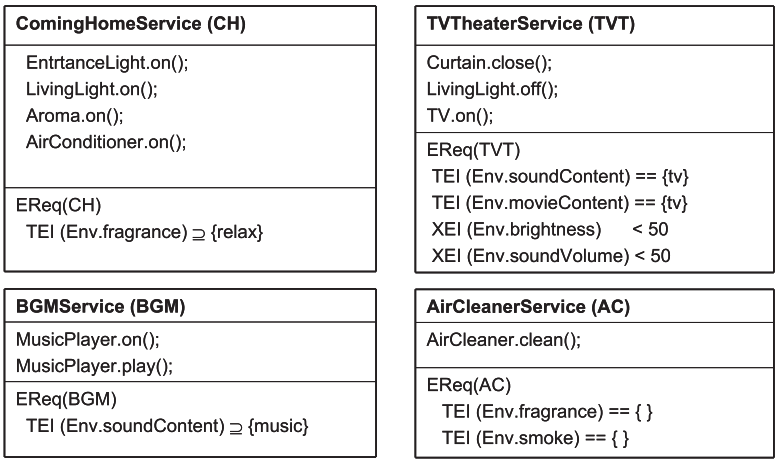


Fig. 5. Integrated services with environment requirements.

Table 1
Result of environment interaction detection ($EnvFI(S_1,S_2)$).

First Service S1	Second Service S2				
	ComingHome (CH)	TVTheater (TVT)	BGM (BGM)	AirCleaning (AC)	Automatic AirCon (AIR)
ComingHome (CH)		∗ ENV-FI (conditional): XEI (Env.brightness) < 50		∗ ENV-FI: TEI (Env.fragrance) ⊇ {relax}	#
TVTheater (TVT)	ENV-FI: XEI (Env.brightness) < 50		∗ ENV-FI: XEI (Env. sound Volume) < 50 TEI (Env.soundContent) == {tv}	∗ ENV-FI: XEI (Env. sound Volume) < 50	
BGM (BGM)		∗ ENV-FI: XEI(Env.soundVolume) < 50 TEI (Env.soundContent) == {tv}		#	
AirCleaning (AC)	∗ ENV-FI: TEI(Env.fragrance) == {}	∗ ENV-FI: XEI (Env. sound Volume) < 50	#		ENV-FI: TEI (Env.temperature) == 25
Automatic AirCon (AIR)	∗ ENV-FI (conditoinal): TEI (Env.temperature) == 25			NV-FI: EI (Env.temperature) = 25	

ment. For instance, we did not explicitly consider what TVT should achieve (or expect) in the environment. Therefore, within the previous definition TVT and any service that affects the same environment property cause an environment interaction. Thus we could not distinguish many acceptable cases from troublesome environment interactions.

It is reasonable to assume that every integrated service S has a (minimal) requirement for the environment that S should achieve and expect. For instance, TVT tries to achieve a theater-like atmosphere for watching a TV, which can be explained by the following requirement: (a) TVT must play the TV content, and (b) the surroundings should be sufficiently dark and peaceful for watching the TV. We call such a requirement the *environment requirement of service*. For convenience we denote $EReq(S)$ to represent the environment requirement of a service S.

Our key idea is to describe $EReq(S)$ by a condition defined over the total and exclusive impacts of S, namely, $TEI(e)$, $TEI(e,S)$ and $XEI(e,S)$. For instance, the above condition (a) of TVT can be described by:

```
TEI(Env.movieContent) == {tv} and
  TEI(Env.soundContent) == {tv}
```

which states that movie and sound content in the room must be only from the TV (== represents an equality). The above (b) can be represented by:

```
XEE(Env.brightness,TVT) < 50 and
  XEE(Env.soundVolume,TVT) < 50
```

Table 2
$Comb(n,k)$: number of combinations to be checked ($\alpha_{n,k} = 0.4$).

	$n = 5$	$n = 10$	$n = 15$	$n = 20$	$n = 25$	$n = 30$
$k = 2$	20	90	210	380	600	870
$k = 3$	24	288	1092	2736	5520	9744
$k = 4$	19	806	5242	18,605	48,576	105,235
$k = 5$	8	1935	23,063	119,071	408,038	1,094,446

meaning that impacts on the brightness and sound volume coming from other services must be minimal. In specifying a condition with an environment impact, operators ==, != (equality), <, <=, >, >= (comparison) are used for the numeric properties. For non-numeric properties we also use set operators like ⊂, ⊆, ⊃, ⊇. We assume that $EReq(S)$ is specified by the home user such that $EReq(S)$ represents what the user desires of S. $EReq(S)$ should not contradict to the functionality of S, however, it does not necessarily cover the full functionality of S.

Fig. 5 shows the four integrated services with their environment requirements. In this example, $EReq(CH)$ represents a modest requirement that the relaxation fragrance should at least be provided. $EReq(TVT)$ is the same as the above. $EReq(BGM)$ represents that BGM wants music to be contained in the environment, although it does not care about other sound content. $EReq(AC)$ states that all smoke and fragrance are expected to be eliminated.

4.2. Environment constraint

Every house usually has a set of rules (i.e. constraints) that the user has to follow in order to assure safe and wholesome living [11]. Such rules include "electric power usage must be less than X kW", or "maximum sound volume should not exceed Y db". In addition to the environment requirements, such rules also have to be satisfied

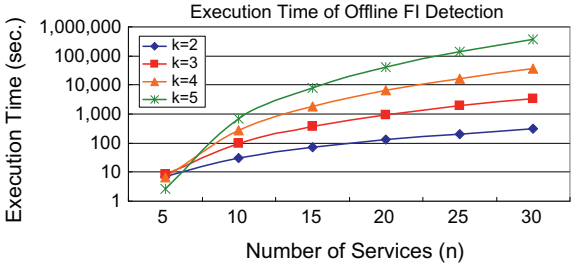


Fig. 6. Execution time of offline detection.

Fig. 7. experimental room of CS27-HNS.

by the integrated services. We call such rules *environment constraints*.

We assume that the environment constraint is a global invariant, denoted by G, and is defined independently of the services or appliances. In this research, we define G as a condition over the total environment impact, $TEI(e)$. For instance, an environment constraint "concurrent electric power usage must be less than 2000 kW" is described as "$TEI(Env.electricity) < 2000$". Each service S must be created to satisfy G, also $EReq(s)$ should be given without violating G.

5. New definition of environment interactions

Based on the environment impacts and the environment requirements, we give a new definition of environment interactions. According to its principle, a feature interaction is an inconsistent conflict among multiple services each of which works correctly [4,12,13]. Therefore, in the context of the HNS, the execution of every service S must satisfy the associated environment requirement $EReq(S)$, as well as the environment constraint G. We represent this condition as follows:

$$S \vdash EReq(s) \wedge G$$

It can be understood that feature interactions occur when the above condition is violated by the execution of multiple services. Let S_1 and S_2 be integrated services, and let $S_1; S_2$ be a sequence of service executions, where S_1 and S_2 are executed in this order. Then, we define the environment interaction between S_1 and S_2, denoted by $EnvFI(S_1, S_2)$, by the following condition:

$$EnvFI(S_1, S_2) \Longleftrightarrow [S_1 \vdash EReq(S_1) \wedge G] \wedge [S_2 \vdash EReq(S_2) \wedge G] \wedge \neg[S_1; S_2 \vdash EReq(S_1) \wedge EReq(S_2) \wedge G]$$

The first line represents that both S_1 and S_2 work normally for the environment requirement and constraint. The second line means that the execution of S_1 and S_2 does not satisfy either $EReq(S_1)$, $EReq(S_2)$ or G. Note in $EnvFI(S_1,S_2)$ that S_1 and S_2 are not commutative, since the condition involves the execution order of S_1 and S_2. We extend the above definition to describe *n-way environment interactions*, $Env(S_1, S_2, \ldots, S_n)$, as follows.

$$EnvFI(S_1, S_2, \ldots, S_n) \Longleftrightarrow \neg EnvFI(S_1, S_2 \ldots, S_i)(i = 2, 3, \ldots, n-1) \wedge \neg[S_1; S_2; \ldots; S_n \vdash EReq(S_1) \wedge \cdots \wedge EReq(S_n) \wedge G]$$

The above condition means that the first $n-1$ services do not cause environment interactions, and that adding an nth service violates the environment requirements or constraint.

The proposed definition $EnvFI(S_1,S_2)$ is basically defined on the sequential execution, $S_1; S_2$, of two services. Let us see what happens if two services are executed *in parallel*, denoted by $S_1 \| S_2$. In $S_1 \| S_2$ the appliance methods of the two services *interleave*. However, the total environment impact caused by $S_1 \| S_2$ is the same as that by $S_1; S_2$, because the sum operation is generally commutative and associative. Therefore, as long as the requirement is violated (i.e. $EReq(S_1) \wedge EReq(S_2) \wedge G$ does not hold) at the end of service execution $EnvFI(S_1,S_2)$ can characterize environment interactions for $S_1 \| S_2$. On the other hand, during $S_1 \| S_2$ suppose that a requirement violation occurs briefly and disappears at the end. Such a *transient* environment interaction is not detected by the proposed method. To capture it precisely we need more powerful logic for $EReq(S_i)$. Although we are optimistic about transient interactions in the sense that they will be resolved at the end, detailed investigation will be left for future work.

6. Case study

6.1. Offline detection of environment interactions

Using the new definition of environment interaction we attempt to detect all the potential environment interactions among the five practical integrated services: Coming Home Service (CH), TV Theater Service (TVT), BGM Service (BGM), Air Cleaning Service (AC), and Automatic Air-Conditioning Service (AIR). The first four services are the same as those introduced in the previous sections. The Automatic Air-Conditioning Service (AIR) automatically controls the air-temperature using an air-conditioner. When a user enters a room the air-conditioner is turned on to a designated temperature (25 °C, for example). In this case $EReq(AIR)$ is `TEI(Env.temperature) == 25`.

To conduct the case study we have implemented a prototype tool, which performs *offline detection* of the environment interactions. The tool was written in Java, comprising about 6300 lines of code, and technical details can be found in [14]. The response time of the tool for checking pair-wise interactions was from 328 ms to 360 ms, using a PC with Intel Core2Duo 2.0 GHz and 960 MB memory.

Table 1 shows the result of pair-wise environment interaction detection. Each row represents a service S_1 that is firstly executed, while the columns represent a service S_2 that follows S_1. Each entry identifies whether or not environment interaction $EnvFI(S_1,S_2)$ occurs, based on the proposed new definition. In the table we omit the appliance interactions due to space limitations. Entries

with ∗ represent environment interactions that could not be reasonably explained by the previous method. Entries with # represent cases that were regarded as environment interactions by the previous method. We explain some of the detected interactions.

ENV-FI (TVT, CH): CH occurs while TVT is being executed. Then the living light and the entrance light are turned on. This increases the brightness of the room, which ruins the theater-like atmosphere required for TVT. This interaction corresponds to FI-2 in Section 1, and can be also characterized by the previous method.

ENV-FI (CH, TVT): TVT occurs after CH. CH does not require the light to be turned on. Hence there is no problem for TVT turning off the living light. The interaction is conditional, depending on the entrance light. If the entrance light does not affect the brightness of the living room by more than 50 lux, there is no interaction. This further explains what was discussed in Limitation 2 in Section 2.3.

ENV-FI (TVT, BGM): While TVT is executed, BGM occurs. The BGM adds `music` to the sound content of the room, which violates *EReq*(*TVT*) that the sound content must be only that of the TV. This interaction corresponds to FI-3 discussed in Limitation 1 in Section 2.3. The interaction is now successfully explained by the proposed method, by introducing the non-cumulative environment property and the environment requirement with the exclusive impact.

ENV-FI (CH, AC): CH is followed by AC. The relaxation fragrance required in CH is removed by AC, which corresponds to FI-4 in Section 1. Introducing an environment impact on the non-cumulative property (based on set operations) can reasonably explain this interaction. Note that the interaction is detected by another condition if we change the execution order. Env-FI (AC, CH) is detected, since adding fragrance by CH violates the condition that the air should be freshened.

There are three cases where the proposed method states no interaction, but the previous method detected it as an interaction. For instance:

CH vs AIR: Both CH and AIR write `Env.temperature`. If the two services have different temperature settings for the air conditioner, the previous method states that this is an environment interaction.[2] However, since CH does not define any requirement on the room temperature, this case should not be an undesirable environment interactions.

6.2. Evaluating scalability

When the number of services increases, the number of service combinations grows combinatorially. For instance, in order to detect pair-wise interactions among 5 services we simply check 20 (= ${}_5P_2$) combinations, as shown in Table 1. For 20 services, however, we have to check 380 combinations. It is too expensive to manually check all of these combinations thus we need an automated tool for managing the FIs among many services.

As mentioned in Section 6.1, we have implemented a tool that conducts the offline detection of environment interactions. We here evaluate the scalability of the tool for a larger number of services. For the evaluation we introduce three kinds of parameters:

- n: the number of all services given.
- k: the number of services to be checked in FI detection.
- $Comb(n,k)$: the number of combinations checked for detecting k-way environment interactions (see Section 5) among n services.

For instance, the case study in Section 6.1 is characterized by a case where $n = 5$, $k = 2$, $Comb(5,2) = 20$. Next let us conduct three-way interaction detection (i.e. $k = 3$). According to the definition, three-way interaction requires a condition that the previous two services cause no interaction. Therefore we first choose 9 pairs out of the 20, which are interaction-free combinations (see Table 1). For each of the 9 pairs we pick a third service from the remaining 3. So we have $Comb(5,3) = 9 * 3 = 27$ in this case.

We try to generalize it. Let $\alpha_{n,k}$ be a ratio of interaction-free combinations out of total $Comb(n,k)$ combinations. Then, we can calculate $Comb(n,k)$ as follows.

$$Comb(n,k) = \begin{cases} {}_nP_2 & (k=2) \\ \alpha_{n,k-1} * Comb(n,k-1) * (n-k+1) & (k \geqslant 3) \end{cases}$$

Although $\alpha_{n,k}$ depends on a given set of services as well as requirements we use an empirical value 0.4, estimated from the previous case study. Table 2 enumerates $Comb(n,k)$ for $(5 \leqslant n \leqslant 30)$, $(2 \leqslant k \leqslant 5)$. As seen in the table, $Comb(n,k)$ grows as n and k increase. The growth is especially significant for the increase in k. Note that n characterizes the number of services deployed in the HNS. From a practical point of view, $n = 30$ is a sufficiently large number of services.

By multiplying $Comb(n,k)$ by the average execution time of the tool we evaluate the scalability of the tool for practical settings. Fig. 6 plots the estimated execution time

[2] If both services share the same air-conditioner they cause an appliance interaction. However this is beyond the scope of this paper.

of the tool. The horizontal axis represents the number of services (n), and the vertical axis represents the execution time in seconds. For pair-wise or three-way FI detection (i.e. $k = 2,3$), our tool can be applied well to 30 services. Even for the case $(n,k) = (30,3)$, the tool performs interaction detection within one hour.

However, for the cases with $k \geqslant 4$ the tool takes quite a long time for many services, due to combinatorial explosion. Therefore for the cases with $k \geqslant 4$, we should consider applying *online* FI detection. That is the condition $EnvFI(S_1, \ldots, S_k)$ is evaluated at runtime, instead of checking all possible combinations of services beforehand.

7. Discussion

7.1. Advantage and limitations

The proposed environment impact model enables the fine-grained modeling of interactions between appliances and the environment. Also the proposed environment requirement explicitly defines what is desired by the service. As a result, the proposed method can more precisely capture the environment interactions in the HNS. As seen in the case study the proposed method was able to detect environment interactions that could not be reasonably explained by the previous method. It could also distinguish some acceptable cases from undesirable interactions.

The principle of the proposed method is to check whether the combined behaviors of multiple services still satisfy all the requirements of the services. In this sense our approach is straight-forward and similar to existing FI methodologies. Although one may consider that it is not state of the art, we believe our contribution adapts the basic principle to environment interactions in HNS.

We are currently developing an efficient detection method for environment feature interactions. This will be integrated into our practical home network testbed CS27-HNS [7], shown in Fig. 7. Using the integrated service manager in Fig. 8, CS27-HNS currently manages appliance interactions and previous environment interactions. The environment interaction with the proposed new definition will be soon available in the service manager.

A limitation is that the proposed method heavily relies upon the given environment impact model and the environment requirements. Therefore an inappropriate impact model may lead to serious interactions being missed, or mis-detection of desirable cases as feature interactions. The same thing holds for the environment requirements. This problem of specifying a consistent system model and properties is also seen in general verification and validation methodologies [11]. How a consistent impact model and requirements are to be described is left for our future research.

There are also issues of *inter-room interactions* and *side-effects*. As mentioned in Section 2.1, we assume that the HNS is defined for every isolated room. Furthermore, we do not count the side-effect of the appliances methods. Moreover, we presume that an integrated service is defined by a simple sequence without loops or branches. An extension to more sophisticated models is also in our future work.

7.2. Related work

Kolberg et al. [4] and Wilson et al. [9] presented runtime feature interaction detection methods in a smart home. In the methods, environment interactions were detected by a condition where different services perform incompatible *resource locking* towards the same environment property. However the methods did not determine the level of the environment impacts or the explicit requirements of the services.

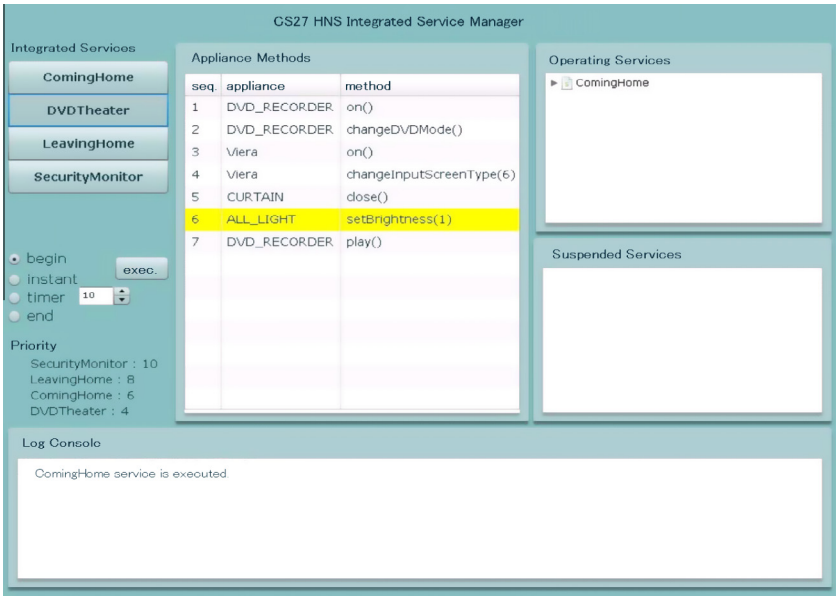


Fig. 8. CS27-HNS integrated service manager.

Metzger et al. [10,15] demonstrated an offline feature interaction detection method for an embedded control system. The method captures dependencies between the system and the environment with a goal-oriented requirement model. However the method basically only managed the static structure of the dependencies and did not consider the degree or the type of dependency, as are considered in our impact model.

Matsuo et al. [8] proposed a model checking method to detect feature interactions in the HNS. However their formalization of the environment interactions only considered the direction of the impact as we discussed in Section 2.3.

The *policy conflict* is a well-studied notion in the field of policy-based system management [16,17]. There are several studies that characterize feature interactions in smart space as policy conflicts (e.g., [18–20]). However these existing methods basically only deal with the action conflicts on the appliances, and do not adequately cover the indirect conflicts on the environment.

In a general sense, the proposed definition of the environment interactions follows the traditional problem frame with the entailment relation ($\vdash$), which was recently discussed with regard to the *context-sharing problem* [21]. In our problem, the shared context is specialized to the environment (i.e. room, entrance and house). Thus the taxonomies based on the context-sharing problem may help us devise context-sensitive detection and resolution schemes for the environment interactions.

There are many studies on *context modeling and reasoning techniques* [22]. They are relevant to the management of environmental contexts within smart spaces. As far as we have investigated, however, there is no existing technique that extensively focuses on the feature interaction on the environment.

8. Conclusion

In this paper we have proposed a new definition of environmental feature interactions in the context of home network services. The proposed method introduces an environment impact model for the fine-grained modeling of interactions between home appliances and the environment. It also considers the environmental requirement of services to explicitly capture the desired behaviors of services towards the environment. It was shown in the case study that the proposed method successfully detects environment interactions that could not be reasonably explained by the previous method.

The following issues are considered for our future work: (a) development of feature interaction resolution for the new environment interactions, (b) investigation of methodologies creating consistent impact models and requirements, (c) extension to more sophisticated models (services with loops and branches, transient interactions, interroom interactions and side-effects), and (d) generalization for other interested domains, such as building control, factory control and smart cities.

Acknowledgments

This research was partially supported by the Japanese Ministry of Education, Science, Sports, and Culture [Grant-in-Aid for Scientific Research (C) (No.24500079), Scientific Research (B) (No.23300009)], and the Kansai Research Foundation for Technology Promotion.

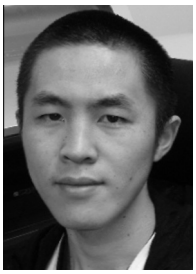
References

[1] Panasonic Corp., Lifinity. <http://www2.panasonic.biz/es/densetsu/lifinity/catalogue/index.html>.
[2] Toshiba Lighting & Technology Corp., Feminity. <http://feminity.toshiba.co.jp/feminity/feminity_eng/index.html>.
[3] M. Calder, E. Magill, M. Kolberg, S. Reiff-Marganiec, Feature interaction: a critical review and considered forecast, Computer Networks 41 (1) (2003) 115–141.
[4] M. Kolberg, E.H. Magill, M. Wilson, Compatibility issues between services supporting networked appliances, IEEE Communications Magazine 41 (11) (2003) 136–147.
[5] M. Nakamura, H. Igaki, K. Matsumoto, Feature interactions in integrated services of networked home appliances – an object-oriented approach –, in: 8th International Conference on Feature Interactions in Telecommunications and Software Systems (ICFI2005), 2005, pp. 236–251.
[6] M. Nakamura, H. Igaki, Y. Yoshimura, K. Ikegami, Considering online feature interaction detection and resolution for integrated services in home network system, in: 10th International Conference on Feature Interactions in Telecommunications and Software Systems (ICFI2009), 2009, pp. 191–206.
[7] M. Nakamura, S. Matsuo, S. Matsumoto, H. Sakamoto, H. Igaki, Application framework for efficient development of sensor as a service for home network system, in: 8th IEEE International Conference on Services Computing (SCC2011), 2011, pp. 576–583.
[8] T. Matsuo, P. Leelaprute, T. Tsuchiya, T. Kikuno, Verifying feature interactions in home network systems, IPSJ Journal 49 (6) (2008) 2129–2143.
[9] M. Wilson, M. Kolberg, E.H. Magill, Considering side effects in service interactions in home automation – an online approach, in: 9th International Conference on Feature Interactions in Telecommunications and Software Systems (ICFI2007), 2007, pp. 172–187.
[10] A. Metzger, S. Buhne, K. Lauenroth, K. Pohl, Considering feature interactions in product lines: towards the autonomic derivation of dependencies between product variants, in: 8th International Conference on Feature Interactions in Telecommunications and Software Systems (ICFI2005), 2005, pp. 198–216.
[11] L. Du-Bousquet, M. Nakamura, B. Yan, 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 5 (3) (2009) 181–196.
[12] A.P. Felty, K.S. Namjoshi, Feature specification and automatic conflict detection, in: 6th International Workshop on Feature Interactions in Telecommunications and Software Systems (FIW2000), 2000, pp. 179–192.
[13] N. Gorse, L. Logrippo, J. Sincennes, Formal detection of feature interactions with logic programming and LOTOS, Journal on Software and System Modeling 5 (2) (2006) 121–134.
[14] K. Ikegami, Formulation and Evaluation of Environment Feature Interactions in Home Network System, Master Thesis, Graduate School of Engneering, Kobe University, 2011. <http://www27.cs.kobe-u.ac.jp/achieve/data/pdf/1124.pdf> (in Japanese).
[15] A. Metzger, Feature interactions in embedded control systems, Journal of Computer Networks 45 (5) (2004) 625–644.
[16] M. Charalambides, P. Flegkas, G. Pavlou, A.K. Bandara, E.C. Lupu, A. Russo, N. Dulav, M. Sloman, J. Rubio-Loyola, Policy conflict analysis for quality of service management, in: Sixth IEEE International Workshop on Policies for Distributed Systems and Networks, 2005, pp. 99–108.
[17] E.C. Lupu, M. Sloman, Conflicts in policy-based distributed systems management, IEEE Transactions on Software Engineering 25 (6) (1999) 852–869.
[18] C. Shankar, R. Campbell, A policy-based management framework for pervasive systems using axiomatized rule-actions, in: Fourth IEEE International Symposium on Network Computing and Applications, 2005, pp. 255–258.

[19] F. Wang, K.J. Turner, Policy conflicts in home care system, in: 9th International Conference on Feature Interactions in Telecommunications and Software Systems (ICFI2007), 2007, pp. 54–65.
[20] N. Khakpour, M. Sirjani, S. Jalili, Formal analysis of smart home policies using compositional verification, in: 10th International Conference on Feature Interactions in Telecommunications and Software Systems (ICFI2009), 2009, pp. 220–233.
[21] A. Nhlabatsi, R. Laney, B. Nuseibeh, Feature interaction as a context sharing problem, in: 10th International Conference on Feature Interactions in Telecommunications and Software Systems (ICFI2009), 2009, pp. 133–148.
[22] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, D. Riboni, A survey of context modelling and reasoning techniques, Journal of Pervasive and Mobile Computing 6 (2) (2010) 161–180.

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 Cybermedia 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, the feature interaction problem and life log. He is a member of the IEEE, IEICE and IPSJ.

Kousuke Ikegami received the B.E. and M.E. degrees in System Informatics from Kobe University, Japan, in 2009 and 2012, respectively. His research interests include home network system and feature interactions.



Shinsuke Matsumoto received the B.E. degree in Physics from Kyoto Sangyo University, Japan in 2006. He received M.E. and Ph.D. degrees in Information Science from Nara Institute of Science and Technology (NAIST) 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 and mining software repository. He is a member of the IEEE and IEICE.