*Feature Interactions in Telecommunications and Software Systems VIII*
*S. Reiff-Marganiec and M.D. Ryan (Eds.)*
*IOS Press, 2005*


# Feature Interactions in Integrated Services of Networked Home Appliances

## — *An Object-Oriented Approach* —

Masahide Nakamura [a,1], Hiroshi Igaki [a] and Ken-ichi Matsumoto [a]
[a] *Nara Institute of Science and Technology, Japan*

**Abstract.** This paper proposes a service-centric framework for the feature interaction problem in integrated services of the home network system (HNS). To formalize the HNS, we extensively use an object-oriented approach where each networked appliance (or the HNS environment) is modeled as an object consisting of properties and methods. Based on the model, we define two types of feature interactions: appliance interactions and environment interactions. An appliance interaction occurs on an appliance object when different services try to invoke methods that perform incompatible updates or references for common properties of the appliance. An environment interaction occurs when methods of different appliances indirectly conflict via the environment object. We conduct a case study of offline interaction detection among several practical service scenarios. It is shown that the proposed framework is quite generic enough to capture the potential interactions in the HNS. We also discuss the feasibility to online detection and several resolution schemes.

**Keywords.** home network, networked appliances, service centric modeling, interaction detection,

## 1. Introduction

Recent advancement in processors, sensors and networks enables emerging technologies to *network* various home electric appliances, including TVs, air-conditioners, lights, DVD players, and refrigerators [2][3][14]. A system consisting of such networked home appliances is generally called a *Home Network System (HNS)*. The HNS provides many applications and services for home users such as group control of appliances [7], health monitoring [17], and home security [11]. Several HNS products have already come onto the market.

A major HNS application is the *integrated service of networked home appliances* (we simply call *HNS integrated service* in the following). The HNS integrated service orchestrates different home appliances to provide more comfortable and convenient living for users, which is considered one of the next-generation value-added services in the ubiquitous computing environment. Typical HNS integrated services include:

**DVD Theater Service:** When a user switches on a DVD player, a TV is turned on in DVD mode, a blind is closed, the brightness of the lights is minimized, 5.1ch

[1]Correspondence to: Masahide Nakamura, Nara Institute of Science and Technology, 8916-5, Ikoma, Takayama, Nara, 630-0101, Japan. Tel.: +81 743 72 5312; Fax: +81 743 72 5319; E-mail: masa-n@is.naist.jp

speakers are selected, and the sound volume of the speaker is automatically adjusted.

**Coming Home Light Service:** When a door sensor notices that the user comes home, lights are automatically turned on. Then, the brightness of the lights are adjusted to an optimal value based on the current degree obtained from an illuminometer.

Feature interactions may occur in HNS integrated services as well, since multiple services may be activated simultaneously. For example, the above two integrated services interact with each other.

**Interactions between DVD Theater & Coming Home Light:** Suppose that a user $A$ activates the DVD Theater service, and simultaneously that a user $B$ comes home. Then, the following two interactions occur:

**FI-(a):** Although the DVD Theater service minimizes a brightness of the lights, the Coming Home Light service sets the brightness comfortable for $B$. This may ruin $A$'s desire to watch the DVD in a comfortable atmosphere.

**FI-(b):** If the blind is closed (by the DVD Theater) immediately after the lights read the degree from the illuminometer (by the Coming Home Light), the lights may fail to set the optimal illumination because the blind makes the room darker.

The feature interaction problem in the HNS integrated services was first addressed by Kolberg et al. [9]. These authors regard each HNS component (an appliance or an environmental variable) as a *resource*. In their model, each integrated service accesses some resources in a *shared* or *not-shared* mode. An interaction is detected when different services try to access a common resource with an incompatible access mode. Thus, each appliance is simply modeled by the two-valued *access attributes*, and each integrated service is characterized only by how the service sets values of the attributes. This simple modeling enables a light weight and realistic implementation framework for feature interaction avoidance (see Section 6.3 for more discussion).

However, the future HNS appliances will have more features, and the HNS integrated services will become more sophisticated and complex. The services may be even customized and personalized by the home users. In such a situation, two slightly different services may yield the same access pattern to the resources, which cannot be differentiated by the conventional method. Hence, we consider it necessary to have a *finer-grained* approach which can reflect features of appliances as well as concrete scenarios of the HNS integrated services.

The goal of this paper is to propose a more *service-centric* framework for feature interactions in the HNS integrated services. In contrast to the previous resource access model, we use an *object-oriented* approach extensively for higher modeling fidelity on the HNS components. Specifically, we model each appliance as an *object* consisting of *properties* and *methods*. The properties characterize the internal states of the appliance, whereas the methods abstract features provided by the appliance. Executing a method may refer or update some properties of the appliance. These dynamics are modeled by a *pre-condition* and a *post-condition*, encapsulating the internal appliances specific implementation of the features. We similarly construct an object for the *home environment*. Then, a HNS is defined by a set of the appliance objects and an environment object. Each HNS integrated service (scenario) is defined as a sequence of the appliance methods.

Within the model, feature interactions are formalized by conflicts among *appliance methods* that are concurrently invoked by different HNS integrated services. We define two types of feature interactions: *appliance interaction* and *environment interaction*. The appliance interaction is a direct conflict between methods $m$ and $m'$ of the same appliance device $d$. It is formulated within $d$ as an *incompatible goal* between the post-conditions of $m$ and $m'$, or, as a *race condition* between the pre-condition of $m$ and the post-condition of $m'$. The above example FI-(a) corresponds to an appliance interaction, where two methods, say, `Light.setBrightness(5)` and `Light.setBrightness(100)` conflict on `Light` object.

On the other hand, the environment interaction is an indirect conflict between methods $m$ and $m'$ of the different appliances $d$ and $d'$, respectively. The conflict occurs via the HNS environment object $e$, when both $m$ and $m'$ write (or $m$ reads and $m'$ writes) a common property of $e$, simultaneously. The above example FI-(b) corresponds to an environment interaction, where two methods, say, `Blind.setGate(close)` and `Illuminometer.getIllumination()` conflict on the `Brightness` property of the environment object.

Based on the formulation, we conduct a practical case study of offline interaction detection. We show that the proposed framework is generic enough to formalize feature interactions in the HNS integrated services. We also discuss the feasibility for online detection and several resolution schemes within the proposed framework.

## 2. Preliminaries

### 2.1. Networked Home Appliances

A HNS consists of one or more *networked appliances* connected to a local area network. In general, each networked appliance has *device control interfaces* by which users or external software agents can control the appliance via a network. For example, every air-conditioner should have interfaces for controlling power and temperature settings. A speaker will have volume and channel (2ch or 5.1ch), etc.

In this paper, we assume that the device control interfaces are provided in the form of *APIs*. Thus, the appliance is supposed to own a processor, a storage (to store device applications or middleware), and a network interface to handle the API calls. This assumption is not unrealistic. Several standards already exist that prescribe a detailed object template for each category of appliances (e.g., [2][3]). Also, the price and size of processors/memories are becoming reasonable enough to embed in home appliances. Some recent products (e.g., [15]) involve a Web application with which the user can configure and control the appliance from external PCs.

The communication among the networked appliances is performed by an underlying protocol. Various protocols for home appliances are proposed, such as X-10 [18], HAVi [6], Jini [8], and UPnP [16]. In this paper, we assume that a certain mechanism (e.g., middleware or gateway) to deal with the underlying protocol is available in the given HNS. Hence, we do not care which underlying protocol should be used to drive the APIs of appliances.

### 2.2. HNS Integrated Services

Controlling only a single networked appliance does not offer much added value compared to traditional appliances [9]. The main advantage of the HNS lies in *integrating* the

control of multiple appliances together, which yield value-added and more powerful *services*. We call such services achieved by the integration of multiple networked appliances *HNS integrated services*.

For a more comprehensive discussion, we introduce an example. In the example, we suppose a HNS consisting of the following ten kinds of appliances (a DVD player, a TV, a speaker, a light, an illuminometer, a door (with a sensor), a telephone, an air-conditioner, a thermometer and a blind). We also assume that one appliance exists for each kind, and that a total of ten appliances are installed in the same room.[1]

We prepare the following seven *service scenarios* of the HNS integrated services (denoted by $SS_i$ $(1 \leq i \leq 7)$). These scenarios are determined based on the actual HNS products [7][11].

$SS_1$: **Auto-TV Service -** When the user turns on the TV, the speaker's channel is set to 2ch, and the volume of the speaker is automatically adjusted for the TV mode.

$SS_2$: **DVD Theater Service -** When a user switches on the DVD player, the TV is turned on in DVD mode, the blind is closed, the brightness of the lights is minimized, the 5.1ch speakers are selected, and the volume of the speaker is automatically adjusted.

$SS_3$: **Coming Home Light Service -** When the door (sensor) notices that the user comes home, the light is automatically turned on. Then, the illumination of the lights are adjusted to the optimal value based on the current degree obtained from the illuminometer.

$SS_4$: **Coming Home Air Conditioning Service -** When the door sensor register that the user has come home, the air-conditioner is turned on, and its temperature setting is adjusted to the optimal based on the current degree of temperature provided by the thermometer.

$SS_5$: **Ringing and Mute Service -** When the telephone rings, the volume of the speaker is muted.

$SS_6$: **Blind Service -** When sunlight is available, the blind is opened.

$SS_7$: **Sleep Service -** When the user goes to bed or goes outside, all appliances are turned off.

Each service scenario can be achieved by executing the APIs of the networked appliances in a certain order. For instance, the above $SS_2$ would be implemented by the following sequence of the API calls. For simplicity, we denote `A.m` to represent the execution of an API $m$ provided by an appliance $A$.

$SS_2$: **DVD Theater Service:**

```
1. DVD.setPower(ON);              /* DVD is turned on. */
2. TV.setPower(ON);               /* TV is turned on. */
3. TV.setInput(DVD);              /* Input mode is set to DVD mode. */
4. Blind.setPower(ON);            /* Blind is turned on. */
5. Blind.setGate(close);          /* Blind is closed. */
6. Light.setPower(ON);            /* Light is turned on. */
7. Light.setBrightness(5);        /* Brightness is minimized. */
```

[1] For multiple appliances in the same kind, we regard them as independent appliances. For example, if there are four lights in the room, we consider four instances; Light1, Light2, Light3 and Light4.

| $SS_1$:Auto-TV | $SS_2$:DVD Theater | $SS_3$:Coming Home Light | $SS_6$:Blind Service |
|---|---|---|---|
| 1.1. TV.setPower(ON)<br>1.2. TV.setInput(TV)<br>1.3. Speaker.setPower(ON)<br>1.4. Speaker.setInput(TV)<br>1.5. Speaker.setChannel(2)<br>1.6. Speaker.setVolume(60) | 2.1. DVD.setPower(ON)<br>2.2. TV.setPower(ON)<br>2.3. TV.setInput(DVD)<br>2.4. Blind.setPower(ON)<br>2.5. Blind.setGate(Close)<br>2.6. Light.setPower(ON)<br>2.7. Light.setBrightness(5)<br>2.8. Speaker.setPower(ON)<br>2.9. Speaker.setInput(DVD)<br>2.10.Speaker.setChannel(5.1)<br>2.11.Speaker.setVolume(80) | 3.1.Door.getDoorStatus()<br>3.2.Illuminometer.setPower(ON)<br>3.3.Illuminometer.getBrightness()<br>3.4.Light.setPower(ON)<br>3.5.Light.setBrightness(600) | 6.1.Blind.setPower(ON)<br>6.2.Blind.setGate(Open) |
| $SS_5$:Ringing and Mute | | $SS_4$:Coming Home Air-Con | $SS_7$:Sleep Service |
| 5.1.Phone.ringing()<br>5.2.Phone.connected()<br>5.3.Speaker.setVolume(30) | | 4.1.Door.getDoorStatus()<br>4.2.Thermometer.setPower(ON)<br>4.3.Thermometer.getTemperature()<br>4.4.AC.setPower(ON)<br>4.5.AC.setTemperature(26) | 7.1.DVD.setPower(OFF)<br>7.2.TV.setPower(OFF)<br>7.3.Speaker.setVolume(0)<br>7.4.Speaker.setPower(OFF)<br>7.5.Illuminometer.setPower(OFF)<br>7.6.Light.setBrightness(0)<br>7.7.Light.setPower(OFF)<br>7.8.AC.setPower(OFF)<br>7.9.Thermometer.setPower(OFF)<br>7.10.Blind.setGate(Close)<br>7.11.Blind.setPower(OFF) |

**Figure 1.** API Sequences for $SS_1$ to $SS_7$

```
8.  Speaker.setPower(ON);          /* Speaker is turned on. */
9.  Speaker.setInput(DVD);         /* Input mode is set to DVD */
10. Speaker.setChannel(5.1);       /* Channel is set to 5.1ch. */
11. Speaker.setVolume(80);         /* Volume is set to 80db. */
```

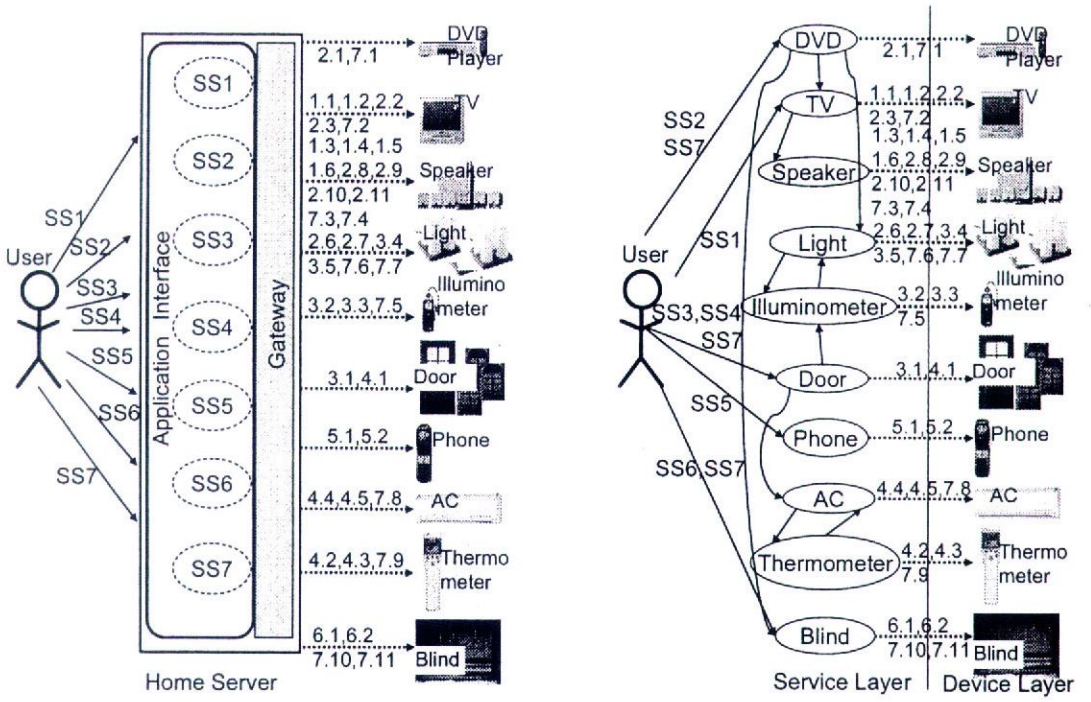
We do not pose any assumptions on *who* designs the HNS integrated services. Thus, the sequence of the API calls could be constructed by appliance vendors, service providers or even home users. Figure 1 shows an example of API sequences for all service scenarios $SS_1$ to $SS_7$.

### 2.3. Architectures for Appliance Orchestration

As seen in the previous subsection, a HNS integrated service can be implemented as a sequence of APIs provided by multiple appliances. To do this, it is necessary for a HNS to have a certain mechanism to *orchestrate* the appliances.

A straightforward way to orchestrate the appliances is to deploy a powerful *home server* in the HNS [7][10][11]. The home server takes centralized control of all appliances in the HNS, which we call *Server Centralized Architecture (SCA)*. Figure 2(a) depicts the SCA-based HNS, implementing the integrated services $SS_1$ to $SS_7$. In the figure, an arrow represents a trigger of a service or an API call indexed by the number in Figure 1. Upon a request from the user, the home server executes the APIs of the appliances in a pre-determined order, which achieves the integrated service requested. The underlying protocols of the appliances may be different from each other (e.g., ECHONET [3] for lights, blinds, and sensors, and a UPnP for Audio/Visual appliances [16]). Therefore, the home server requires a sophisticated gateway [14] to achieve the interoperability among the appliances.

Delegating the appliance control to the appliances themselves is another way to have appliance orchestration. In [13], we proposed an autonomous-decentralized architecture based on the *Service Oriented Architecture* (SOA) [10], which is shown in Figure 2(b). Attaching an application adaptor (called a *service layer*, depicted by an oval) to each appliance, an appliance can autonomously trigger other appliances, with a generic protocol (such as XML/SOAP), for a given service scenario. This architecture requires no centralized home server, which improves the reliability, flexibility, and scalability of the HNS. Note that whatever architecture is taken for the appliance orchestration, the HNS inte-




(a) server-centralized (b) autonomous-decentralized (with SOA)

**Figure 2.** HNS Architectures for Appliance Orchestration

grated service can be implemented basically as a *sequence of API calls* of the appliances. This research proposes a generic framework *independent* of the HNS architecture.

## 3. Formal Definition of HNS

### 3.1. Model of Appliance

Each appliance can be regarded as an *object* consisting of *properties* (also called *attributes*) and *methods*. The properties characterize the current status of the appliance. On the other hand, the methods represent public application interfaces through which some properties are referred or updated from outside [3] [2]. In this paper, the methods correspond to the APIs discussed in Section 2.1. For instance, every appliance has a property `Power` whose value is basically either `ON` or `OFF`. An air-conditioner generally has a property `TemperatureSetting` by which the air-conditioner produces air with an appropriate temperature. The air-conditioner may have a method (thus, an API), `setTemperature()`, by which the user or an external software agent can update the current value of `TemperatureSetting`. Each property has a *type* to define an allowable range of the property value. For a property $Prop$, we denote $tProp$ to represent the type of $Prop$. We assume that for each appliance, properties with corresponding types and methods are given by the vendor of the appliance (e.g., with a manual).

Table 1 summarizes an example of properties and types for the ten appliances introduced in Section 2.2. Due to limited space, properties irrelevant to the $SS_1$ to $SS_7$ are omitted from the table. For example, the air-conditioner has properties `Power` and

[2] Several standardizations of the appliance object model are currently under way. For example, ECHONET prescribes detailed properties required for each appliance class.

**Table 1.** Appliance Properties

| Appliance | Property | PropertyType |
|---|---|---|
| AirConditioner | Power | {ON,OFF} |
| | TemperatureSetting | unsigned int (°C) |
| Thermometer | Power | {ON,OFF} |
| | CurrentTemperature | unsigned int (°C) |
| Speaker | Power | {ON,OFF} |
| | Input | {TV,DVD} |
| | Channel | {2,5.1} |
| | VolumeSetting | unsigned int (dB) |
| Light | Power | {ON,OFF} |
| | BrightnessSetting | unsigned int (lx) |
| Illuminometer | Power | {ON,OFF} |
| | CurrentBrightness | unsigned int (lx) |
| Door | DoorStatus | {Open,Close} |
| | Power | {ON,OFF} |
| Phone | PhoneStatus | {Received, Calling, Connected, Waiting} |
| DVD player | Power | {ON,OFF} |
| TV | Power | {ON,OFF} |
| | Input | {TV,DVD} |
| Blind | Power | {ON,OFF} |
| | BlindStatus | {Open,Close} |

`TemperatureSetting`, where `tPower = {ON, OFF}` and `tTemperatureSetting = unsigned int`. Also, the air-conditioner may implement methods such as `setPower (tPower onoff)` and `setTemperature(tTemperatureSetting temp)`. The air-conditioner is controlled from the network by executing the methods with parameters such as: `setPower(ON)` and `setTemperature(25)`.

The details of each method are usually encapsulated in an appliance-specific feature implementation. Therefore, several abstraction levels can be considered to model the method. In this paper, for the generality of the model, we simply characterize each method as a pair of *pre-conditions* and *post-conditions*. The pre-condition is a condition required *before* the execution of the method, while the post-condition is a condition that holds *after* the method is executed.

We specify each pre(or post)-condition by a *property formula* constructed with some properties of the appliance. For example, consider the method `setTemperature(tTemperatureSetting temp)`. Suppose that the implementation of this method is as follows: "When the power is on, if the method is executed, the temperature is set to the value specified by `temp`". Then the method can be specified with a pre-condition:`Power == 'ON'` and a post-condition: `TemperatureSetting == temp`. More generally, we specify each pre(or post)-condition as a *conjunction* of Boolean formulas with properties.

**Definition 3.1 (Property Formula)** Let $P = \{p_1, p_2, ..., p_n\}$ be a given set of properties. A formula $c = f_{p_1} \wedge f_{p_2} \wedge ... \wedge f_{p_n}$, where $f_{pi}$ is any Boolean formula with respect to $p_i$, is called a *property formula* over $P$. $Cond_P$ denotes a set of all property formulas over $P$. For $c = f_{p_1} \wedge f_{p_2} \wedge ... \wedge f_{p_n}$, $\prod_{p_i}(c) = f_{p_i}$ is called a *projection* of $c$ with respect to property $p_i$.

Let us consider the example of the air-conditioner. In this case then, a formula $c = [$`Power=='ON'` $\wedge$ `TemperatureSetting` $> 20]$ is a property formula, which is supposed to become true when the power is on and the value of the temperature setting is greater than 20 degree. Note that $c$ is a conjunction of Boolean formulas each of which depends on only a single property. Also, $\prod_{Power}(c) =$ [ `Power=='ON'` ], which is a projection of $c$ onto `Power`. Next, we define each networked appliance as follows.

**Table 2.** Appliance Models

| Appliance | Method | Pre-Condition | Post-Condition |
|---|---|---|---|
| AirConditioner | setPower(tPower onoff) | | Power=onoff |
| | setTemperature(tTemperature temp) | Power='ON' | TemperatureSetting=temp |
| Thermometer | setPower(tPower onoff) | | Power=onoff |
| | getTemperature() | Power='ON' ∧ CurrentTemperature=* | |
| Speaker | setPower(tPower onoff) | | Power=onoff |
| | setInput(tInput spInput) | Power='ON' | Input=spInput |
| | setChannel(tChannel spChannel) | Power='ON' | Channel=spChannel |
| | setVolume(tVolume spVolume) | Power='ON' | VolumeSetting=spVolume |
| TV | setPower(tPower onoff) | | Power=onoff |
| | setInput(tInput tvInput) | Power='ON' | Input=tvInput |
| DVD | setPower(tPower onoff) | | Power=onoff |
| Light | setPower(tPower onoff) | | Power=onoff |
| | setBrightness(tBrightness lx) | Power='ON' | BrightnessSetting=lx |
| Illuminometer | setPower(tPower onoff) | | Power=onoff |
| | getBrightness() | Power='ON' ∧ CurrentBrightness=* | |
| Door | getDoorStatus() | Power='ON' ∧ DoorStatus=* | |
| Phone | ringing() | PhoneStatus='Recieved' | PhoneStatus='Calling' |
| | connected() | PhoneStatus='Calling' | PhoneStatus='Connected' |
| Blind | setPower(tPower onoff) | | Power=onoff |
| | setGate(tGate gateStatus) | Power='ON' | BlindStatus=gateStatus |

**Definition 3.2 (Networked Home Appliance)** A *networked home appliance* $d$ is defined as a quad tuple $d = (P_d, M_d, Pre_d, Post_d)$, where

- $P_d$ is a set of all *properties* of $d$.
- $M_d$ is a set of all *methods* of $d$.
- $Pre_d$ is a *pre-condition* function $M_d \rightarrow Cond_{P_d}$, which maps each method $m \in M_d$ into a property formula. $m$ can be executed only when $Pre_d(m)$ is true.
- $Post_d$ is a *post-condition* function $M_d \rightarrow Cond_{P_d}$, which maps each method $m \in M_d$ into a property formula. $Post_d(m)$ becomes true immediately after $m$ is executed.

To avoid confusion, a method $m \in M_d$ of an appliance $d$ is denoted by $d.m$.

Table 2 shows a simplified model of all the appliances in our example. In the table, '`*`' denotes a *don't care* value. For example, the condition `CurrentTemperature == *` in $Pre_d$ of `Thermometer.getTemperature()` becomes true, as long as a certain value of the property is available.

### 3.2. Environment

Each appliance deployed in a HNS shares a home space with other appliances. Therefore, the appliances are tightly coupled with the *environment* of the home. For instance, the air-conditioner tries to keep a comfortable room temperature, which implicitly updates the temperature of the environment. Also, the thermometer refers to the current temperature of the environment. Thus, the air-conditioner and the thermometer are indirectly connected via the environment, which can impact the comfortableness of the HNS users.

Thus, the environment of the home is an important factor in feature interaction analysis (cf. [9][12]). In this paper, we formalize the environment as a *global object* which can be referred to or updated by all appliances in the HNS. Specifically, an environment object has a set of *global properties* such as temperature, brightness and sound volume.

When a method $m$ of an appliance is executed, these environment properties are indirectly referred to or updated by $m$. For the environment, we adopt a loose modeling such that we only care whether the method $m$ *reads* or *writes* some environment properties or not. This model is because the impact of a method to the environment properties

**Table 3.** Environment Model

| Appliance | Method | Re | We |
|---|---|---|---|
| AirConditioner | setPower() | | |
| | setTemperature() | | Temperature |
| Thermometer | setPower() | | |
| | getTemperature() | Temperature | |
| Speaker | setPower() | | |
| | setInput() | | |
| | setChannel() | | |
| | setVolume() | | Volume |
| TV | setPower() | | |
| | setInput() | | |
| DVD | setPower() | | |
| Light | setPower() | | |
| | setBrightness() | | Brightness |
| Illuminometer | setPower() | | |
| | getBrightness() | Brightness | |
| Door | getDoorStatus() | | |
| Phone | ringing() | | Volume |
| | connected() | | Volume |
| Blind | setPower() | | |
| | setGate() | | Brightness,Temperature |

are not as direct and explicit as the impact to the appliance properties [3]. Therefore, we cannot specify strict pre/post conditions with the environment properties before/after the execution of $m$.

**Definition 3.3 (Environment)** Let $D = \{d_1, d_2, ..., d_k\}$ be a set of all appliances deployed in the HNS. Also, let $M = \cup_{d_i \in D} M_{d_i}$ be a set of all methods of all appliances. Then, an environment $e$ is defined as a tuple $e = (P_e, R_e, W_e)$, where

- $P_e$ is a set of all environment properties.
- $R_e$ is an *environment read* function $M \rightarrow 2^{P_e}$, which maps each method $m \in M$ into a set of environment properties that are read by $m$.
- $W_e$ is the *environment write* function $M \rightarrow 2^{P_e}$, which maps each method $m \in M$ into a set of environment properties that are written by $m$.

In our example HNS, we assume the following environment properties:

`Temperature`: the current degree of temperature of the room.
`Brightness`: the current intensity of brightness in the room.
`Volume`: the current sound volume in the room.

Table 3 shows an environment model for our example HNS. The columns $R_e$ and $W_e$, respectively, show which environment properties are read or written by each appliance method. For example, environment property `Temperature` is designated in $W_e$(`AirConditioner.set Temperature(...)`). This property implies that setting the temperature of air-conditioner can write (update) the current temperature degree of the home.

### *3.3. HNS and Integrated Services*

We are now ready to formalize the HNS. The HNS consists of a set of appliances deployed and an environment.

**Definition 3.4 (Home Network System)** A home network system is defined as $HNS = (D, e)$, where

[3] For instance, the temperature setting of an air-conditioner is not always equal to the temperature of a room.

- $D = \{d_1, d_2, ..., d_n\}$ is a set of appliances.
- $e = (P_e, R_e, W_e)$ is an environment where the HNS is deployed.

Therefore, our example HNS consists of the ten appliances defined in Tables 1 and 2 and an environment defined in Table 3. Next, as mentioned in Section 2.2, we assume that a HNS integrated service is given by a *scenario* without branches. Specifically, we define the service as a sequence of appliance methods.

**Definition 3.5 (HNS Integrated Service Scenario)** Let $HNS = (D, e)$ be a given HNS. Then, a sequence of any appliance methods $ss_i = d_{i1}.m_{i1}, d_{i2}.m_{i2}, ..., d_{ik}.m_{ik}$ $(d_{ij} \in D, m_{ij} \in M_{d_{ij}})$ is called an *HNS integrated service scenario*.

Thus, the API sequences shown in Figure 1 are finally formalized as the HNS integrated service scenarios.

## 4. Feature Interactions in the HNS Integrated Services

If multiple integrated service scenarios are executed in a HNS, unexpected conflicts between the scenarios may occur. In this paper, we propose two kinds of feature interactions for the HNS integrated services, specifically *appliance interactions* and *environment interactions*.

### *4.1. Appliance Interactions*

When multiple service scenarios simultaneously invoke incompatible methods of a common appliance, one method conflicts with another, which results in a feature interaction on the appliance. We formalize the conflict among methods on the same appliance as *appliance interactions*.

Let us consider $SS_1$ and $SS_2$ in Figure 1 as an example. $SS_1$ invokes `Speaker.setChannel(2)`, while $SS_2$ invokes `Speaker.setChannel(5.1)`. Hence, if $SS_1$ and $SS_2$ are simultaneously executed, a race condition occurs in which channel 2 or 5.1 should be set to the speaker. According to Table 2, the simultaneous execution of $SS_1$ and $SS_2$ updates the value of the property `Channel` of the Speaker into two different values `2` and `5.1`. This situation is characterized by two *unsatisfiable post-conditions* on the common appliance property `Channel`; `[Channel==2]` $\wedge$ `[Channel==5.1]` $= \bot$ (unsatisfiable). Similarly, $SS_1$ and $SS_2$ cause an appliance interaction on the property `Input` of the TV.

Let us introduce another example with $SS_1$ and $SS_7$. `TV.setInput(TV)` of $SS_1$ requires in the pre-condition that the TV is switched on (`[power==ON]`). However, `TV.Power(OFF)` of $SS_7$ updates the value of the property `power` into `OFF` as defined in its post-condition, which disables `TV.setInput(TV)` of $SS_1$. This situation can be explained by the fact that a pre-condition and a post-condition are unsatisfiable simultaneously.

From the above observations, we define the appliance interactions as conflicts among methods on a common property of an appliance.

**Definition 4.1 (Appliance Interactions)** Let $HNS = (D, e)$ be a given HNS, and $ss_i$ and $ss_j$ be a pair of integrated service scenarios defined on $HNS$. Suppose that for an

appliance $d \in D$, $ss_i$ contains a method $d.m_i$ and $ss_j$ contains a method $d.m_j$. We say that $ss_i$ and $ss_j$ cause an *appliance interaction* on $d$ iff at least one of the following conditions is satisfied:

**Condition D1:** There exists an appliance property $p \in P_d$ such that $\prod_p Post(m_i) \wedge \prod_p Post(m_j) = \perp$ (unsatisfiable), or

**Condition D2:** There exists an appliance property $p \in P_d$ such that $\prod_p Post(m_i) \wedge \prod_p Pre(m_j) = \perp$ (unsatisfiable).

### 4.2. Environment Interactions

The environment interaction refers an indirect conflict among appliances via the HNS environment. This interaction arises when different appliance methods try to access common environment properties at the same time. Note that the methods causing the interaction are not always executed in the same appliance.

**Definition 4.2 (Environment Interactions)** Let $HNS = (D, e)$ be a given HNS, and $ss_i$ and $ss_j$ be a pair of integrated service scenarios defined on $HNS$. Suppose that for a pair of appliances $d, d' \in D$ $(d \neq d')$, $ss_i$ contains a method $d.m_i$ and $ss_j$ contains a method $d'.m_j$. We say that $ss_i$ and $ss_j$ cause an *environment interaction* iff at least one of the following conditions is satisfied:

**Condition E1:** $W_e(m_i) \cap W_e(m_j) \neq \phi$, or

**Condition E2:** $R_e(m_i) \cap W_e(m_j) \neq \phi$.

Condition E1 reflects a race condition between two 'writes' on the common environment properties. Condition E2 specifies non-interchangeable 'read' and 'write' on the common environment properties.

For example, suppose that $SS_3$ and $SS_6$ in Figure 1 are executed simultaneously. In $SS_3$, the light must be optimally adjusted based on the illuminometer. On the other hand, $SS_6$ opens the blind, which ruins the optimal light adjustment. This situation can be explained as follows. As shown in Table 2, `Light.setBrightness()` of $SS_3$ and `Blind.setGate()` of $SS_6$ try to write the common environment property `Brightness`. Moreover, `Illuminometer.get Brightness()` of $SS_3$ reads `Brightness` as well. That is, $W_e$(`Light.setBrightness()`) $\cap$ $W_e$(`Blind.setGate()`) $\cap$ $R_e$(`Illuminometer.getBrightness()`) = {`Brightness`} $\neq \phi$. Thus, these three methods cause an environment interaction on brightness in the home.

## 5. Case Study: Offline Interaction Detection

We have conducted a case study of an offline interaction detection. For this experiment, we have implemented a tool. The tool takes a specification of a HNS based on the proposed framework, and detects all possible interactions in the specification. The case study here is formulated as follows:

**Offline feature interaction detection**

**Input:** A home network system $HNS = (D, e)$ specified in Tables 1, 2 and 3. A set of HNS integrated service scenarios $SS_1, SS_2, ..., SS_7$ shown in Figure 1.

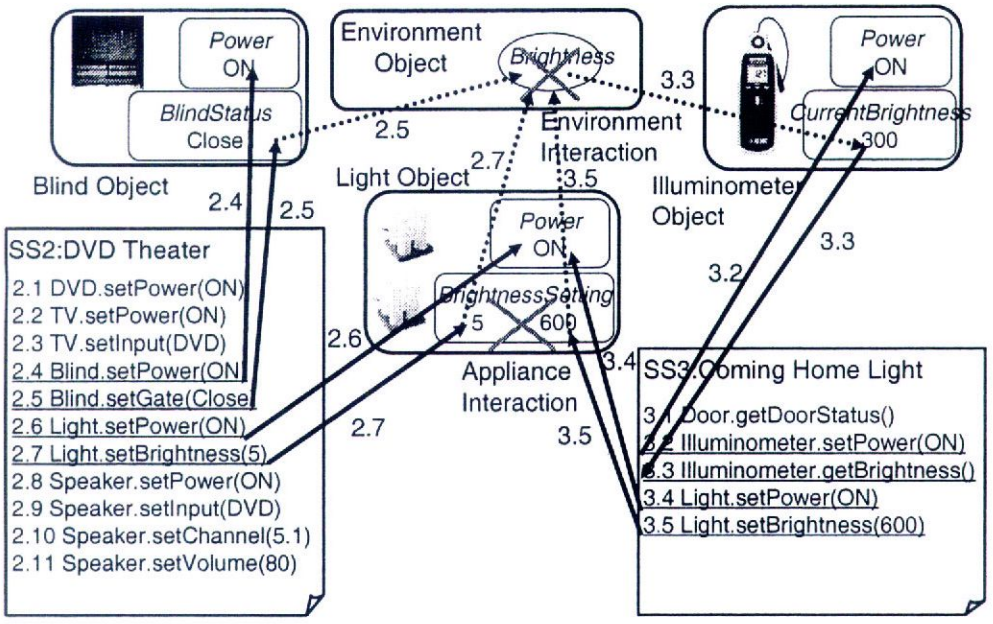



**Figure 3.** Interactions between $SS_2$ and $SS_3$

**Output:** All possible pairs of appliance methods that cause appliance or environment interactions.

**Procedure:** For any pair of methods $m$ and $m'$ contained in $SS_i$ and $SS_j$, respectively, evaluate Conditions D1 and D2 for appliance interactions, and Conditions E1 and E2 for environment interactions.

Table 4(a) shows a total 43 appliance interactions, whereas Table 4(b) enumerates 24 environment interactions. Each entry represents a set of pairs of conflicting methods.

For example, let us look at feature interactions between $SS_2$ (DVD Theater) and $SS_3$ (Coming Home Light). Figure 3 depicts the detailed scenario of the interactions showing how each method in a service updates or refers a property of an appliance (shown as a solid arrow), or indirectly accesses the environment object (shown as a dotted arrow) [4]. $SS_2$ and $SS_3$ cause an appliance interaction on the `Light`. Specifically, methods 2.7 and 3.5 conflict on the property `BrightnessSetting`, since both methods try to modify the property in different ways. This interaction is detected by Condition D1 (see Definition 4.1), which is exactly the same way that FI-(a) was introduced in Section 1. The services also cause an environment interaction, where the methods 2.5 and 3.3 make a race-condition between read and write of the environment property `Brightness`. The interaction is detected by Condition E2 (see Definition 4.1), by which we can reasonably explain FI-(b) as explained in Section 1. Two other environment interactions occur as well between 2.5 and 3.5 and between 2.7 and 3.3.

## 6. Discussion

In this section, we discuss several important issues for the feature interaction problem in the HNS integrated services. Due to limited space, we present only a sketch for each issue. The detailed methodologies and implementations are left to our future publication.

[4] Objects and properties that are not related to the interaction scenario are omitted from the figure.

**Table 4.** Results of the Offline Interaction Detection

(a) Appliance Interactions

| | SS1 | SS2 | SS3 | SS4 | SS5 | SS6 | SS7 |
|---|---|---|---|---|---|---|---|
| SS1 | | (1.2,2.3)(1.4,2.9) (1.5,2.10)(1.6,2.11) | | | (1.6,5.3) | | (1.1,7.2)(1.2,7.2)(1.3,7.4) (1.4,7.4)(1.5,7.4)(1.6,7.3)(1.6,7.4) |
| SS2 | | | (2.7,3.5) | | (2.11,5.3) | (2.5,6.2) | (2.1,7.1)(2.2,7.2)(2.3,7.2)(2.4,7.9) (2.5,7.8)(2.5,7.9)(2.6,7.7)(2.7,7.6) (2.7,7.7)(2.8,7.4)(2.9,7.4)(2.10,7.4) (2.11,7.3)(2.11,7.4) |
| SS3 | | | | | | | (3.2,7.5)(3.3,7.5)(3.4,7.7)(3.5,7.6) (3.5,7.7) |
| SS4 | | | | | | | (4.2,7.9)(4.3,7.9)(4.4,7.8)(4.5,7.8) |
| SS5 | | | | | | | (5.3,7.3)(5.3,7.4) |
| SS6 | | | | | | | (6.1,7.11)(6.2,7.10)(6.2,7.11) |
| SS7 | | | | | | | |

(b) Environment Interactions

| | SS1 | SS2 | SS3 | SS4 | SS5 | SS6 | SS7 |
|---|---|---|---|---|---|---|---|
| SS1 | | | | | (1.6,5.1)(1.6,5.2) | | |
| SS2 | | | (2.5,3.3)(2.5,3.5)(2.7,3.3 | (2.5,4.3)(2.5,4.5 | (2.11,5.1)(2.11,5.2) | (2.7,6.2) | (2.5,7.6)(2.7,7.10) |
| SS3 | | | | | | (3.3,6.2)(3.5,6.2 | (3.3,7.6)(3.3,7.10)(3.5,7.10) |
| SS4 | | | | | | (4.3,6.2)(4.5,6.2 | (4.3,7.10)(4.5,7.10) |
| SS5 | | | | | | | (5.1,7.3)(5.2,7.3) |
| SS6 | | | | | | | (6.2,7.6) |
| SS7 | | | | | | | |

### 6.1. *Feasibility to Runtime Interaction Detection*

As seen in Section 5, the offline interaction detection shows all *potential* interactions among HNS integrated service scenarios. When a pair of service scenarios $ss_i$ and $ss_j$ are shown to cause a feature interaction (by the offline detection), the safest solution is to *uninstall* either $ss_i$ or $ss_j$ from the HNS. However, this solution would significantly limit flexible creation and deployment of the service scenarios. Preferably, the interaction should be managed *during runtime* only when it occurs. We here evaluate the feasibility of the proposed framework to the runtime (online) interaction detection. The feasibility is discussed by architecture-wise examination with the server-centralized architecture (SCA, see Figure 2(a)) or the autonomous-decentralized architecture with SOA (SOA, see Figure 2(b)).

In the SCA, detecting both appliance and environment interactions can be undertaken by the home server. The home server takes the global control of all appliances. Therefore it can monitor the current values of all properties of every appliance, as well as an environment object. That is, the home server can track a *global state* of the HNS. Thus, every time the home server invokes a method in a service, the server evaluates Conditions D1, D2, E1 and E2 to detect appliance and environment interactions.

On the other hand, the SOA requires a sophisticated mechanism for the runtime detection, because of its fully-distributed appliance control. Detecting the appliance interactions during runtime is not very hard, since evaluation of Conditions D1 and D2 can be done locally within a single appliance. The appliance interactions can be detected by installing an application, say *FI detector*, for each appliance. The FI detector continuously monitors all properties of the appliance. Then, it warns an interaction (by Conditions D1 and D2) when multiple services try to invoke conflicting methods. A smart implementation of the FI detector could be to utilize an *aspect-oriented approach* [1], which captures

any accesses from multiple arbitrary methods to a common property as a cross-cutting concern.

In the SOA, detecting the environment interactions is possible but is a bit harder without having a global server simulating the environment object. To achieve this, each appliance first communicates with every appliance causing potential environment interactions, using the result of offline detection. If any method conflicting on the environment is being executed, then the appliance reports an environment interaction.

### 6.2. Resolution of Feature Interaction

Once a feature interaction is detected in a pair of HNS integrated services, it should be *resolved*. For this, several approaches can be considered.

**(a) Rebuild Scenario:** Based on the result of the offline detection, rebuild the service scenarios so that any interaction is avoided. This approach is available only in the environment where the services can be flexibly rebuilt.

**(b) Prompt User:** When an interaction is detected during runtime, ask the home user(s) to determine manually how the interaction should be dealt with. A user probably would be more comfortable if the prompt message is delivered with appropriate choices of resolution, e.g., (Choice 1:) Give up the whole service execution, (Choice 2:) Compromise some functionalities (i.e, methods) of the service, or (Choice 3:) Automatically retry later.

**(c) Prioritize Services:** Assign static priorities to services [9]. If a pair of service scenarios cause an interaction, then all conflicting methods in the service with lower priority are aborted. Note in this approach that the execution of the service with a higher priority is always guaranteed.

**(d) Prioritize Methods:** Assign static priorities to methods. When a pair of methods conflict with each other, methods with a lower priority are aborted. Although functionalities of the services are partially limited, both conflicting services may be run through without being aborted.

**(e) Prioritize Users:** Assign static priorities to users. A user with a higher priority can take precedence in executing services over the one with a lower priority.

**(f) Compromise Services:** Find a compromise between the conflicting services during runtime. In the service scenario, set a weight of *importance* to each method. For example, methods related to the DVD player, the TV, and the speaker are important for DVD Theater service, but the ones for the light and the blind may be auxiliary. When an interaction occurs, the service is compromised so that at least important methods are executed while auxiliary methods are aborted.

**(g) Compromise Methods:** Find a compromise between the conflicting methods during runtime. For example, a conflict between `Speaker.setVolume(50)` and `Speaker.setVolume(10)` would be compromised to `Speaker.setVolume(30)`. If there is no good compromise, then assign *dynamic* priorities to the methods to abort one of them. The priority could be derived based on, for example, the frequency of the service execution, elapsed time of the service. For more complex cases, sophisticated schemes should be prepared in advance based on the result of the offline detection.

**(h) Negotiate Among Users:** Find a solution acceptable for users by conducting a negotiation. This approach is quite realistic as it is usually done manually in our daily life. A smarter approach will involve user agents which perform an automatic negotiation and resolution based on the user's policy and/or preference.

As shown above, many granularity can be considered for interaction resolution. Using multiple resolution schemes together achieves fine interaction management for each of system, service, method and user levels. Especially, the resolution schemes at the method level ((d) and (g)) take full advantage of the proposed service-centric framework. Indeed, more approaches for the interaction resolution within the HNS integrated service domain may exist. Further discussion on the interaction resolution schemes is left to our future research.

### *6.3. Related Work*

Compared to the existing method [9], the proposed method enables finer-grained interaction analysis and resolution on the concrete service scenarios. However, the proposed method contains some similar parts, which can be regarded as a *specialization* of conventional methods. For example, Condition D1 and E1 can characterize Kolberg's MAI and STI interactions respectively, in a more detailed level of abstraction. Also, Conditions D2 and E2 cover SAI and MTI. Thus, the proposed method can be used to implement their runtime interaction avoidance [9], by converting each appliance method into a nameless action or trigger and gathering appropriately the appliance properties into the two-valued access attributes.

As far as is reported, explicit consideration of the environmental factor in the control application was first introduced by Metzger [12]. Within the domain of embedded control systems, their approach captures the static structures of requirements and systems by dependency graphs, and conducts offline interaction detection for systems under development. Our method differs in targeting the HNS where the appliances and services can be dynamically added and modified. Hence, the proposed framework took into careful consideration the *modularity*. Specifically, all features provided by a device (appliance) should be encapsulated a self-contained object, which is *loosely coupled* with other objects.

We are also investigating the conventional techniques in the telephony domain. We found some techniques quite promising for implementation using the proposed method. For example, the approaches with logic programming (e.g., [4]) and/or structural analysis of rule-based methods (e.g., [19]) would enable efficient pre/post-conditions checking of the appliance methods. A negotiating agent approach [5] would also help to implement an automatic interaction resolution for the scheme (h) in Section 6.2.

## 7. Conclusion

In this paper, we have presented a service-centric framework for feature interactions in the HNS integrated services. First, we proposed a formal model of the HNS in an object-oriented fashion. Within the model, two types of feature interactions were defined. A feature interaction occurs on an appliance object or an environment object when multiple methods in different services try to refer/update common properties of the object. The interactions were formalized as unsatisfiable pre/post conditions. We conducted a case study of offline interaction detection among concrete service scenarios. Also, topics on the online detection and several resolution schemes were discussed.

Several research directions present themselves. We are currently implementing concrete methodologies for online detection and resolution with the proposed framework.

Especially important is evaluating the feasibility of the suggested resolution schemes from several viewpoints; system-view, service-view, and user-view and so forth. Adaptation of the conventional techniques in telephony to the HNS integrated services is also interesting topic for further study.

## Acknowledgment

This research was partially supported by: the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Young Scientists (B) (No.15700058), and Grant-in-Aid for 21st century COE Research (NAIST-IS — Ubiquitous Networked Media Computing).

## References

[1] L. Blair and J. Pang, "Aspect-oriented solutions to feature interaction concerns using AspectJ", *Proc. of Seventh Int'l. Workshop on Feature Interactions in Telecommunication Networks and Distributed Systems (FIW'03)*, pp.87-104, Sep. 2003.
[2] Digital Living Network Alliance - `http://www.dlna.org`
[3] ECHONET Consortium - `http://www.echonet.gr.jp/`
[4] N. Gorse, "The feature interaction problem: Automatic filtering of incoherences & generation of validation test suites at the design stage", Master's Thesis, University of Ottawa, Ottawa, Ontario, Canada, 2001.
[5] N. D. Griffeth and H. Velthuijsen, "The Negotiating Agents Approach to Runtime Feature Interaction Resolution", *Proc. Second Int'l Workshop Feature Interactions in Telecommunications Systems*, pp. 217-235, 1994.
[6] HAVi - `http://www.havi.org/`
[7] Hitachi Home & Life Solutions inc., "horaso network" - `http://ns.horaso.com/`
[8] Jini - `http://www.jini.org/`
[9] M. Kolberg, E. H. Magill, and M. Wilson, "Compatibility issues between services supporting networked appliances", *IEEE Communications Magazine*, vol. 41, no. 11, Nov 2003 pp. 136-147
[10] S. W. Loke, "Service-oriented device ecology workflows", *Proc. of 1st Int'l Conf. on Service-Oriented Computing (ICSOC2003)*, LNCS2910, pp.559-574, Dec. 2003.
[11] Matsushita Electric Industrial Co., Ltd., Kurashi net `http://national.jp/appliance/product/kur ashi-net/`
[12] A. Metzger, C. Webel, "Feature interaction detection in building control systems by means of a formal product model", *Proc. of Feature Interaction in Telecommunications and Software Systems VII*, pp.105-121, 2003.
[13] M. Nakamura, H. Igaki, H. Tamada and K. Matsumoto, "Implementing integrated services of networked home appliances using service oriented architecture", *Proc. of 2nd International Conference on Service Oriented Computing (ICSOC2004)*, pp.269-278, Nov. 2004.
[14] OSGi Alliance - `http://www.osgi.org/`
[15] Toshiba Corporation, "net de navi" - `http://www.rd-style.com/`
[16] UPnP Forum - `http://www.upnp.org/`
[17] T. Tamura, T. Togawa, M. Ogawa, and M. Yoda, "Fully automated health monitoring system in the home," *Med. Eng. Phys.*, Vol. 20, No. 8, pp. 573-579, 1998.
[18] X-10 - `http://www.x10pro.com/`
[19] T. Yoneda and T. Ohta, "A formal approach for definition and detection of feature interactions", *Proc. of Fifth Workshop on Feature Interactions and Software Systems (FIW'98)*, pp.165-171, IOS Press 1998.