On Detecting Service Chains in Sensor-Driven Home Network Services

Takuya Inada, Kosuke Ikegami, Shinsuke Matsumoto, Masahide Nakamura and Hiroshi Igaki
Graduate School of System Informatics, Kobe University
1-1 Rokkodai-cho, Nada-ku, Kobe, Hyogo 657-8501, Japan
Email: {inada, ikegami}@ws.cs.kobe-u.ac.jp, {shinsuke, masa-n, igaki}@cs.kobe-u.ac.jp

***Abstract*—When multiple sensor-driven services are deployed in the same environment, execution of a service may trigger other services, successively. Such chain reactions of services often cause undesirable feature interactions. This paper presents a framework that can characterize and detect the service chains within the home network system (HNS). We first introduce the ECA rules to describe the services, and then propose an environment effect model to capture how each device in the HNS affects the environment. Finally, we develop an algorithm that detects the service chains with concrete enabling conditions. A case study with 7 practical services shows that the proposed method successfully detects 11 service chains, in which 6 harmful feature interactions are identified.**

***Keywords*-smart home; home network system; sensor-driven service; feature interactions; detection; validation;**

I. Introduction

Research and development of *home network system(HNS)* draw great attention as a key technology of the next-generation smart home [1][2]. In the HNS, house-hold appliances (e.g., TV, DVD Recorder, air-conditioner, lamp, fan) and equipments (e.g., curtain, ventilator, ceiling light) are integrated via a network to achieve various value-added services. Moreover, introducing sensors (e.g., temperature, brightness, motion, electricity, touch) to the HNS can provide autonomous *sensor-driven services*. A sensor-driven service is triggered automatically, depending on a designated *context* characterized by sensor values. Examples of the sensor-driven services are listed as follows:

DVD Theater Service (DVD-T): This service allows a user to watch a movie in a theater-like atmosphere. When the user touches a touch sensor, a TV is turned on, a light is dimmed, a curtain is closed, and a DVD recorder is played.

Automatic Light Control Service (ALC): This service automatically turns on a light when the room becomes dark. When the brightness is lower than 200lx and the user is in the room, a light is automatically turned on.

Automatic Room Heating Service (ARH): This service automatically heats the room with an air-conditioner. When the room temperature is colder than 14°C, the air-conditioner is turned on with a heating mode of 28°C.

Energy Saving Air-Conditioning Service (ESAC): This service controls an air-conditioner for energy saving. When the total electricity exceeds 1500Wm and the air-conditioner is working in a heating mode, the temperature of the air-conditioner is adjusted to 25°C.

These sensor-driven services work fine when they are used separately. However, when multiple services are deployed in the same environment, execution of a service successively triggers another service in a certain condition. We define this phenomena as *service chain*, representing a chain reaction of sensor-driven services within the HNS. The service chain often causes undesirable conflicts among services, known as the *feature interaction problem*[3][4]. For the above four services, we can observe the following service chains.

Service Chain C1 (DVD-T⇒ALC): When a user runs DVD-T, a light is diminished and the room becomes dark. As a result, the condition "brightness is lower than 200lx" holds. So, ALC is triggered and a light is turned on. This ruins the theater-like atmosphere of DVD-T.

Service Chain C2 (ARH⇒ESAC): Suppose that the room temperature drops below 14°C, and that ARH starts the air-conditioner with the temperature setting 28°C to heat the room. In course of time, the electricity consumption rises greater than 1500Wm. As a result, ESAC is triggered and the temperature setting is adjusted to 25°C. Thus, the temperature setting of ARH is overwritten by ESAC.

Service Chain C3 (ALC⇒ESAC): Suppose that room brightness drops below 200lx, and that ALC turns on a light. If the total electricity becomes greater than 1500Wm at this time, ESAC is automatically triggered.

In the above scenarios, the service chains C1 and C2 cause undesirable or unexpected behaviors for the users, whereas the service chain C3 causes nothing wrong. Thus, not all service chains are harmful, but they are potential factors of feature interactions. So, when deploying many sensor-driven services in the HNS, all service chains must be identified in advance, and appropriate resolution must be taken.

The goal of this paper is to propose a framework that can characterize and detect all potential service chains for given sensor-driven services within the HNS. Specifically, we first introduce the *ECA (event, condition, action) rules* to describe the sensor-driven services. We then develop the *environment effect model* to capture explicitly how each appliance affects the environment. Finally, we develop the *service chain detection algorithm* that can derive concrete conditions describing when the service chain occurs.

We conducted a case study with 7 practical services operated in our HNS. As a result, we could detect 11 service chains, in which 6 harmful feature interactions are identified.

978-1-4244-9528-3/12/$26.00 ©2012 IEEE

II. Preliminaries

A. Home Network System

A home network system (HNS) consists of multiple *networked appliances* connected to the local area network at home. Each networked appliance has control APIs, with which users or software agents can control the appliance via the network. Each appliance is generally equipped with a network adapter, a processor and a storage.

A HNS typically involves a *home server*, which manages all the appliances deployed. It also plays a role of *gateway* to the external network. More importantly, various value-added *services* are installed in the home server. When a service is requested, the home server executes APIs of the appliances according to the logic specified in the service. As seen in Section I, a service can orchestrate different appliances at the same time. In our research group, we are building an actual HNS (CS27-HNS) using legacy home appliances [5].

B. Sensor-driven Service in HNS

Introducing sensors to the HNS can make the services autonomous and context-aware. Since the sensors can capture events and contexts, we use them as triggers of the service. In this paper, we write *sensor-driven service* to represent any HNS service triggered by a sensor. For example, let us take Automatic Light Control Service (ALC) in Section I. The condition "the brightness is lower than 200lx" can be evaluated by a light sensor. Thus, ALC can be implemented as a program which invokes API `Light.on()` when the reading of the light sensor becomes less than 200.

The sensor-driven services are smart and convenient in the sense that they do not require human operations. However, if many services are deployed in the same environment, they often cause unexpected service chains, as seen in Section I. Since the number of potential service chains grows combinatorially in the number of services, we need a systematic method to detect and resolve the service chains.

C. Previous Work: Feature Interactions in HNS

In our previous work [3][4], we proposed an object-oriented modeling method to formalize and detect feature interactions in the HNS. In this method, we modeled every appliance (or an environment) as an *object* with *properties* and *methods*. The properties characterize the internal state of the appliance (or the environment), while the methods correspond to the API. Also, we defined every service as a sequence $m_1(); m_2(); ...; m_n()$ of the appliance methods. Then, we defined that a feature interaction between services s_1 and s_2 occurs if a method $m()$ of s_1 and another method $m'()$ of s_2 conflict, either locally on an appliance object (called, *appliance interaction*) or indirectly via an environment object (called, *environment interaction*). However, this method assumed that every service is triggered *manually* by the user. Thus, the sensor-driven services and the incidental service chains were beyond the scope.

III. Modeling Sensor-Driven Services and Service Chains

A. Describing Sensor-Driven Services with ECA Rules

In order to capture the nature of the sensor-driven services, we here introduce the *ECA (event, condition, action) rules* [6] for the service description. A *sensor-driven service* S is defined by $S = (E_S, C_S, A_S)$ where

- E_S is an *event* that triggers S, which is defined by a condition over a *single* environment property.
- C_S is an *enabling condition* to determine the execution of S. On detecting E_S, S is actually executed only when C_S is satisfied. C_S is defined by a condition over appliance properties and environment properties.
- A_S is an *action* to be executed, which is defined by a sequence $m_1(); m_2(); ...; m_n()$ of appliance methods.

This model newly involves the event and condition, compared to the one in the previous work. Figure 1 shows service descriptions of seven sensor-driven services, including the 4 services seen in Section I and 3 more described below.

Automatic Room Cooling (ARC): This service automatically cools down the room, using an air-conditioner. When the room temperature is warmer than 25°C, the air-conditioner is turned on with a cooling mode of 23°C.

Leaving Home (LH): This service shuts down all appliances when a user leaves home. When the user touches a button in the entrance, a TV, a DVD player, an air-conditioner and a light are turned off, and a curtain is closed.

Energy Saving in Absence (ESIA): This service automatically turns off appliances for energy saving in user's absence. When a sensor detects that nobody in the room, the service turns off a TV, a DVD player, an air-conditioner and a light.

In Figure 1, `env` denotes a prefix of an environment property. Each event (or condition) is supposed to be detected (or evaluated, respectively) by appropriate sensors in the HNS. In each action, `A.m()` denotes a method `m()` of an appliance `A`. Let us take the description of ALC. The event of ALC is `env.brightness < 200`, specifying that the service is triggered when the brightness is less that 200lx. The condition `env.absence == false` means that the service should be enabled only when somebody is in the room. If ALC is executed, the light is turned on with brightness level 10 as specified in the action `Light.setBrightness(10)`.

B. Modeling Environmental Effects of Appliances

To detect the service chains, we have to know how much *effect* is given to the environment as a result of a service. Such effect is produced by appliance methods executed as an action of the service. For example, `Light.on()` increases `env.brightness`, and `Air-Conditioner.cooling()` decreases `env.temperature`. Therefore, we propose an *environment effect model* for each appliance, to define explicitly how the appliance gives the environmental effects.

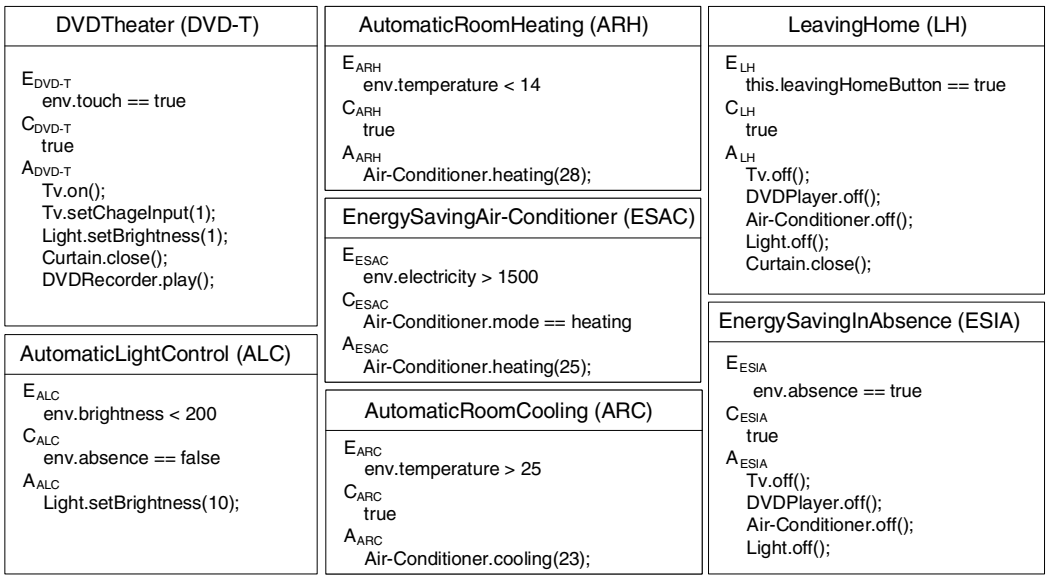

Figure 1. Service description of sensor-driven services with ECA rules

As shown in Section II-C, every appliance can be regarded as an object with internal states. Therefore, we model every appliance as an FSM (finite state machine) specifying the effects within transitions. Let d be an appliance. The *environment effect model* of d is defined by an FSM $EM_d = (S_d, M_d, T_d, s_0, e_d)$, where

- S_d is a set of states of d.
- M_d is a set of appliance methods of d.
- $T_d : S_d \times M_d \rightarrow S_d$ is a state transition function.
- $s_0 \in S_d$ is the initial state.
- e_d is an environment effect function, associating each transition $t \in T_d$ with a set of expressions over environment properties.

Tables I shows the environment effect models of an air-conditioner, a TV and a light, respectively. Each table describes an FSM in a table form, where a row represent a state, a column represents a method. Each entry represents a state transition, containing the effects to the environment and the next state (labeled by `next`). In the effects, =, += and -= respectively represent substitution, addition and subtraction operators. For example, Table I(b) represents that the TV has two states OFF and ON. If method on() is executed within OFF, the state moves to ON. At this time, as the environment effects, the electricity is increased by 500Wm, and the brightness is increased by 200lx.

For some environment properties, the effect may be given gradually. For example, it takes time for an air-conditioner to change the room temperature in the designated value. However, in this paper we define the environment effect by an expected value converged after sufficient time.

We assume that every appliance d in the HNS is first in the initial state of EM_d. When a service $S = (E_S, C_S, A_S)$ is executed, each method $d.m()$ in A_S is executed one by one. Then, a corresponding transition t in EM_d occurs and environment effects $e(t)$ are accumulated to the environment.

Table I
ENVIRONMENT EFFECT MODEL

(a) Air-Conditioner

State \ Method	off()	heating(setTemp)	cooling(setTemp)
OFF	next : OFF	env.electricity += 1500 env.temperature = setTemp next : HEATING	env.electricity += 1500 env.temperature = setTemp next : COOLING
HEATING	env.electricity -= 1500 env.temperature = 10 next : OFF	env.temperature = setTemp next : HEATING	env.temperature = setTemp next : COOLING
COOLING	env.electricity -= 1500 env.temperature = 10 next : OFF	env.temperature = setTemp next : HEATING	env.temperature = setTemp next : COOLING

(b) TV

State \ Method	off()	on()
OFF	next : OFF	env.electricity += 500 env.brightness += 200 next : ON
ON	electricity -= 500 env.brightness -= 200 next : OFF	next : ON

(c) Light

State \ Method	off()	on()	setBrightness(setB)
OFF	next : OFF	env.brightness += 100*this.bLevel env.electricity += 50 next : ON	env.brightness += 100*setB env.electricity += 50 this.bLevel = setB next : ON
ON	env.brightness -= 100*this.bLevel env.electricity -= 50 next : OFF	next : ON	env.brightness += 100*(setB-this.bLevel) this.bLevel = setB next : ON

C. Service Chain

A service chain occurs when the result of one service triggers another service, successively. We try to formalize this mechanism using the proposed service description and

the environment effect model. Let $S_A = (E_{S_A}, C_{S_A}, A_{S_A})$ and $S_B = (E_{S_B}, C_{S_B}, A_{S_B})$ be two services. A *service chain* from S_A to S_B, denoted by $S_A \Rightarrow S_B$, occurs when the environment effects produced in A_{S_A} create an environment (state) where E_{S_B} is satisfied. A chain $S_A \Rightarrow S_B$ may cause another chain $S_B \Rightarrow S_C$, creating $S_A \Rightarrow S_B \Rightarrow S_C$. Also, if S_B triggers S_A again, then a loop $S_A \Rightarrow S_B \Rightarrow S_A \Rightarrow S_B \Rightarrow ...$ may be produced.

Note that the occurrence of $S_A \Rightarrow S_B$ is *conditional*. That is, it depends on the current states of the appliances and the environment. For example, for ALC⇒ESAC in Section I, if no other appliance is working before the execution of ALC, ESAC may not start. Therefore, it is important to identify the *pre-condition* of the service chain, explaining when the chain occurs. We denote $[cond]S_A \Rightarrow S_B$ to represent that a chain $S_A \Rightarrow S_B$ occurs when a pre-condition $cond$ holds.

IV. Detecting Service Chains among Sensor-driven Services

A. Service Chains Detection Algorithm

Now we present an algorithm that can automatically detects the service chains among sensor driven services, using the proposed service description and the environment effect model. Specifically, for a given pair of services $S_A = (E_{S_A}, C_{S_A}, A_{S_A})$ and $S_B = (E_{S_B}, C_{S_B}, A_{S_B})$, the algorithm checks if $S_A \Rightarrow S_B$ occurs. Moreover, if the chain occurs, the algorithm derives a concrete pre-condition $cond$ such that $[cond]S_A \Rightarrow S_B$ holds. Intuitively, the algorithm checks if the environment effects produced by A_{S_A} may create a situation where E_{S_B} is satisfied, as discussed in Section III-C. It consists of the following nine steps.

Step1: Pick out an environment property p in E_{S_B}.

Step2: By analyzing A_{S_A}, obtain a set $M_p = \{m_1(), m_2(), ..., m_n()\} \subseteq A_{S_A}$ of appliance methods where every $m_i()$ has an environment effect on p.

Step3: For every $m_i() \in M_p$, analyze the environment effect models and obtain a set T_{m_i} of any state transitions on which $m_i()$ appears.

Step4: Construct a Cartesian product $T_p = T_{m_1} \times T_{m_2} \times ... \times T_{m_n}$. Each element $tt = (t_1, t_2, ..., t_n) \in T_p$ represents a sequence of transitions each of which has an environment effect on p.

Step5: For each $tt = (t_1, t_2, ..., t_n) \in T_p$, calculate the *total environment effect* $TE(tt) = \Sigma\ [e_p(t_1), e_p(t_2), ..., e_p(t_n)]$, where $e_p(t_i)$ is an environment effect added to the environment property p by the transition t_i. The semantics of Σ will be defined later.

Step6: Let s_e be any environment state where S_B is enabled (i.e., E_{S_B} holds), and let s_d be any state where S_B is disabled (i.e., $\neg E_{S_B}$ holds). For each $tt \in T_p$, if there exists such a pair (s_d, s_e) that $TE(tt)$ changes the state s_d into s_e, then detect a service chain $S_A \Rightarrow S_B$. For this, we call tt *potential transition sequence of the service chain*.

Step7: If $tt = (t_1, t_2, ..., t_n) \in T_p$ is a potential transition sequence of the service chain, for each t_i $(1 \leq i \leq n)$, derive a condition $dcond_i$ of an appliance where t_i is executed. Then, make a conjunction $dcond_p = dcond_1 \wedge dcond_2 \wedge ... \wedge dcond_n$, which is a pre-condition of the service chain w.r.t. the appliances. In addition, obtain a condition $econd_p$ on the environment where s_d of Step 6 is satisfied, which is a pre-condition of the service chain w.r.t. the environment.

Step8: Validate that $cond_p = dcond_p \wedge econd_p$ does not contradict to $E_{S_A} \wedge C_{S_A}$. If there is no contradiction, derive $cond_p$ as the pre-condition of $S_A \Rightarrow S_B$. If there is a contradiction, S_A is disabled. So conclude that $S_A \Rightarrow S_B$ does not occur. (End)

Let v_i be the value of the environment effect $e_p(t_i)$. The semantics of Σ in Step5 is defined by one of the following S1, S2 or S3, depending on the environment property p interested.

S1(Arithmetic Sum): $\Sigma\ [e_p(t_1), e_p(t_2), ..., e_p(t_n)] = v_1 + v_2 + ... + v_n$. This semantics is taken when p is a *cumulative* property, including brightness, sound volume, etc.

S2(Maximum Value): $\Sigma\ [e_p(t_1), e_p(t_2), ..., e_p(t_n)] = max(v_1, v_2, ..., v_n)$. This semantics is taken when the value of p is overwritten by the maximum one, including the temperature of heater, wind level, etc.

S3(Minimum Value): $\Sigma\ [e_p(t_1), e_p(t_2), ..., e_p(t_n)] = min(v_1, v_2, ..., v_n)$. This semantics is taken when the value of p is converged to the minimum one, including the temperature of air-conditioner.

We briefly summarize the purpose of each step of the algorithm. Step1 finds the key property p by which the service chain $S_A \Rightarrow S_B$ may occur. Step2 extracts appliance methods that potentially give an effect to p. Step3 identifies all state transitions related to the methods within the environment effect models. Step4 derives all possible combinations of the transitions affecting p. Step5 calculates the total impact to p produced by each of the combinations. Step6 examines if the total impact can trigger E_{S_B} or not. Step7 derives the pre-condition of the service chain. Finally, Step8 validates if the pre-condition does not contradicts to the enabling condition of S_A.

B. Running Example

Using the proposed algorithm, we demonstrate to detect the service chain C1 (DVD-T⇒ALC) in Section I. In this example, we use the service description in Figure 1, and the environment effect model of the appliances in Table I.

Detection of Service Chain (DVD-T ⇒ ALC)

In Step1, we pick up `env.brightness` from E_{ALC} and know that the brightness is the key property of the service chain. In Step2, we obtain methods `TV.on()`, `Light.setbrightness(1)` from A_{DVD-T}, since they have effects on `env.brightness`. In Step3, we identify the corresponding transitions in the environment

effect models. In Table I(b), we find $T_{TV.on()} = \{$(OFF,on(),ON),(ON,on(),ON)$\}$. In Table I(c), we find $T_{Light.setbrightness(1)} = \{$(OFF, setBrightness(1), ON), (ON, setBrightness(1), ON)$\}$. Step4 constructs a product $T_{TV.on()} \times T_{Light.setbrightness(1)}$ to derive the following four transition sequences.

- tt_1 = ((OFF,on(),ON),(OFF,setBrightness(1),ON))
- tt_2 = ((OFF,on(),ON),(ON,setBrightness(1),ON))
- tt_3 = ((ON,on(),ON),(OFF,setBrightness(1),ON))
- tt_4 = ((ON,on(),ON),(ON,setBrightness(1),ON))

Step5 calculates the total the environment effect to env.brightness. Since the brightness is cumulative, we follow the Semantics S1.

- $TE(tt_1)$ =+200 + 100 = +300
- $TE(tt_2)$ =+200 +100(1-bLevel)=300-100*bLevel
- $TE(tt_3)$ =+0+100 = +100
- $TE(tt_4)$ =+0+100(1-bLevel)=100-100*bLevel

In Step6, let s_d be any state where ALC is not triggered, i.e., $\neg E_{ALC}$ = env.brightness $\geq$ 200 holds. Also, let s_e be any state where ALC is triggered, i.e., E_{ALC} = env.brightness $<$ 200 holds. Among tt_1 to tt_4, we want to find any transition sequences that can change s_d to s_e. Since both tt_1 and tt_3 give positive impact to the brightness, they cannot move s_d to s_e. So, tt_2 and tt_4 are chosen as the potential transition sequences of the service chain.

In Step7, we first derive the pre-condition of the service chain from tt_2. In order for the sequence tt_2 to be executed, the TV must be OFF and the light is ON beforehand. So, $dcond_{bright}$ = [TV is OFF $\wedge$ Light is ON]. From $\neg E_{ALC}$, we obtain a condition env.brightness $\geq$ 200 to be satisfied *before* DVD-T. Also, as for the condition *after* DVD-T, we have env.brightness $+TE(tt_2) <$ 200 so that ALC is triggered. Therefore, $econd_{bright}$ = [200 $\leq$ env.brightness $<$ 100*bLevel$-$100]. Similarly, we derive the pre-condition from tt_4. As a result, we can see that the service chain DVD-T⇒ALC occurs when DVD-T is executed under the one of the following pre-conditions.

- cond1: TV is OFF, Light is ON and a condition [200 $\leq$ env.brightness $<$ 100*bLevel$-$100] holds, or
- cond2: TV is ON, Light is ON and a condition [200 $\leq$ env.brightness $<$ 100*bLevel$+$100] holds.

These pre-conditions are validated in Step8 that they do not disable DVD-T. So, we finally detect [cond1] DVD-T ⇒ ALC and [cond2] DVD-T ⇒ ALC.

V. Case Study

To evaluate the proposed method, we have conducted a case study to detect all service chains among the seven sensor-driven services shown in Figure 1. For the case study, we have implemented a *service chain detection tool*.

Table II shows the result. In the table, a row represents a service triggered first (called first service, say S_A), and a column represents one triggered second (called second service, say S_B). Each entry shows whether a service chain $S_A \Rightarrow S_B$ occurs or not. Out of all $42 (= 7 \times 6)$ possible combinations, 11 service chains were automatically detected by the tool. By investigating the service chains detected, we have found that 6 cases cause undesirable feature interactions (marked as FI in Table II). Scenarios of DVD-T⇒ALC and ARH⇒ESAC are the same as C1 and C2 in Section I, respectively. Scenarios of other FIs are explained below.

Table II
Service chain detection result

First Service S_A \ Second Service S_B	DVD-T	ALC	ARH	ESAC	ARC	LH	ESIA
DVD-T		Chain FI		Chain			
ALC				Chain			
ARH				Chain FI	Chain FI		
ESAC			Chain				
ARC				Chain			
LH		Chain FI	Chain FI				
ESIA		Chain	Chain FI				

Service Chain Scenario (ESIA⇒ARH)

In a day of winter, as nobody remains the room, ESIA turns off all the appliances. This situation makes the temperature decrease, and ARH is triggered in due course. The service chain violates the goal of energy-saving of ESIA.

Service Chain Scenario (LH⇒ARH)

This service chain is similar to ESIA⇒ARH. As a user presses the button of LH, all the appliances are shut down. This makes the temperature decrease and has ARH triggered. This service chain is against the user's requirement that all the appliances should be off while leaving home.

Service Chain Scenario (ARH⇒ARC)

As the temperature decreases, ARH is triggered to heat the room using the air-conditioner. This makes room warm enough to trigger ARC to cool the room, which violates the goal of ARH. The algorithm derives a pre-condition that the air-conditioner is OFF or working with the temperature setting 25 °C or less.

Service Chain Scenario (LH⇒ALC)

If a user presses the button of LH in the night, all the appliances are shut down, and the room becomes dark. Then, ALC is triggered and the light is turned on again. The chain violates the user's requirement that all the appliances should be off while leaving home. The algorithm derives a pre-condition that [200 $\leq$ env.brightness $<$ 200 + 100$*$ Light.bLevel, the light is ON, and TV is OFF].

VI. Discussion

A. Advantage and Limitations

For given sensor-driven services, the proposed method can detect all potential service chains automatically. It also derives concrete pre-conditions for every service chain detected. Therefore, the proposed method can make service developers aware of unexpected service chains in advance. In this sense, the proposed method is useful especially for the offline validation at the design stage of the services.

A limitation is that the proposed algorithm does not yet conclude which of the service chains are actually harmful. As mentioned before, not all service chains lead to undesirable feature interactions. We are currently developing a method that evaluates *severity* of the service chain, based on the gap between user's expectation and actual environment state. Developing reasonable *resolution schemes* of such fatal chains is also an important issue to be tackled.

B. Related Work

Kolberg et al. [7] presented a classification of feature interactions in smart home, where our service chains can be categorized as the *sequential action interaction* (SAI). However, the concrete detection method of the service chains was not described. Wilson et al. [8] took the environment effects of the appliances for detecting feature interactions. However, since the amount of the effects was not explicitly considered, it would be difficult to derive detailed pre-conditions of the service chains. The ECA rules have been often used for detecting policy conflicts [6][9][10]. These methods basically focused on action conflicts only, and the service chains were not explicitly considered. In [11], we proposed a service-oriented framework that facilitates development of individual sensor-driven services. However, this method did not cover interactions among multiple services.

VII. Conclusion

In this paper, we have proposed a method that detects service chains among sensor-driven services in the home network system (HNS). The proposed method adopts the ECA rules for the service description, and introduces the environment effect model to define the effect of the appliances to the environment, explicitly. We also implemented the proposed method as a tool, and detected 11 service chains among 7 practical services. We are currently developing a method that evaluates the severity of the detected service chains. Based on this, we plan to propose systematic resolution schemes of undesirable service chains. Extending the method for general sensor services outside the HNS is also a challenging issue.

References

[1] Panasonic Electric Works Co., Ltd. Lifinity. http://denko.panasonic.biz/Ebox/kahs/

[2] Toshiba. Toshiba Network Appliance Feminity. http://www3.toshiba.co.jp/feminity/about/index.html

[3] H. Igaki and M. Nakamura. Modeling and Detecting Feature Interactions among Integrated Services of Home Network Systems. in IEICE Transactions on Information and Systems, vol.E93-D, no.4, pp.822-833, April 2010.

[4] M. Nakamura, H. Igaki, Y. Yoshimura and 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), pp.191-206, June 2009.

[5] M. Nakamura, A. Tanaka, H. Igaki, H. Tamada, and K. ichi Matsumoto. Constructing Home Network Systems and Integrated Services Using Legacy Home Appliances and Web Services. in International Journal of Web Services Research, pp.82-98, January 2008.

[6] F. Wang and K. J. Turner. Policy Conflicts in Home Care System. in 9th International Conference on Feature Interactions in Telecommunications and Software Systems (ICFI2008), pp.54-65, June 2009.

[7] M. Kolberg, E. Magill, and M. Wilson. Compatibility Issues between Services Supporting Networked Appliances. in IEEE Communications Magazine, vol.41, no.11, pp.136-147, Nov 2003.

[8] M. Wilson, M. Kolberg, and E. H. Magill. Considering Side Effects in Service Interactions in Home Automation - an Online Approach. in Proc. Int'l. Conf. on Feature Interactions in Software and Communication Systems (ICFI'07), pp.172-187, 2007.

[9] A.F. Layouni, K.J. Turner and L. Logrippo. Conflict Detection in Call Control Using First-Order Logic Model Checking. in 9th Feature Interactions in Telecommunications and Software Systems, pp.66-82, 2007.

[10] M.H.T. Beek, S. Gnesi, C. Montangero, and L. Semini. Detecting policy conflicts by model checking UML state machines. in ICFIIOS Press (2009), pp.59-74, 2009.

[11] M. Nakamura, S. Matsuo, S. Matsumoto, H. Sakamoto, and H. Igaki. Application Framework for Efficient Development of Sensor as a Service for Home Network System. in 8th IEEE 2011 International Conference on Services Computing (SCC 2011), pp.576-583, July 2011. (Washington D.C.)