Detecting Feature Interactions in Home Appliance Networks

Pattara Leelaprute\textsuperscript{1}, Takafumi Matsuo\textsuperscript{2}, Tatsuhiro Tsuchiya\textsuperscript{2} and Tohru Kikuno\textsuperscript{2}
\textsuperscript{1}Department of Computer Engineering, Faculty of Engineering, Kasetsart University, Thailand
pattara.l@ku.ac.th
\textsuperscript{2}Graduate School of Information Science and Technology, Osaka University, Japan
\{t-matuo, t-tutiya, kikuno\}@ist.osaka-u.ac.jp

Abstract

Home appliance networks now have the capability of integrating different features of independent appliances to provide value-added services. Concurrent execution of these services, however, can cause unexpected problems, even when each service is independently correct. This paper addresses the issue of detecting such interactions between services. We propose an approach that consists of two steps. In the first step, a model is developed to capture the behavior of the services and the interactions between them and users. In the second step, the model is automatically analyzed to see if possible interactions exist. This automatic analysis can be effectively performed with model checking techniques. The usefulness of the proposed approach is demonstrated through a case study, where several interactions were successfully detected.

Keyword: Home Appliance Network, Model Checking, SPIN, Feature Interaction

1 Introduction

As home appliances are becoming increasingly interconnected, the use of home appliance networks is being expanded \[4, 12, 14, 16\]. Home appliance networks integrate different features of appliances to provide value-added services. For example, by integrating an air-conditioner, a ventilator and thermometers, one can implement an energy-saving HVAC (heating, ventilation and air-conditioning) service. Another example could be an air-cleaning service which automatically cleans the room air by controlling a ventilator and a smoke sensor.

Concurrent execution of these services, however, can cause unexpected behaviors of the system, even when each service is independently correct. This problem can also arise between the above two services. Assume that the HVAC service is operating the air-conditioner to warm up the room temperature and at the same time the air-cleaning service is using the ventilator to clean the room air. If the room temperature is higher than the outside temperature, then cool outside air is taken into the room by the ventilator, which results in the low efficiency of the HVAC service. This problem is called feature interaction \[5\].

In general, design flaws in concurrent systems are hard to find because it is extremely difficult to test all concurrent states. To cope with this difficulty, we focus on model checking \[3\]. Model checking allows an automatic and exhaustive verification of software and system designs modeled as finite state machines. The correctness criteria are specified in a temporal logic. When the design fails to meet the correctness criteria, model checking tools can usually produce a counterexample. Using this counterexample, one can easily detect the cause of the error.

In this paper, we propose a method for detecting feature interactions by means of the SPIN model checker \[7\]. SPIN is one of the most powerful model checkers supporting Linear-Time Temporal Logic (LTL) \[15\]. The main contributions of this paper are as follows:

- We propose a method for describing a home appliance network and its user in Promela, the input language for SPIN. Our proposed approach focuses only on the high-level behavior of services,
and thus can be used independently of underlying network protocols, such as [4, 6, 16].

- We classify feature interactions based on their causes. We also devise an LTL formula for each type of feature interaction.

2 Preliminaries

2.1 Home Appliance Networks

An example of a home appliance network is shown in Figure 1. The home appliance network consists of an air-conditioner, a ventilator, a smoke sensor, two thermometers (inside and outside a room), a DVD player, a TV set, lights, a curtain blind, an illuminometer and a home server.

2.2 Services in Home Appliance Networks

By integrating features of different appliances, convenient services can be implemented. These value-added services are one of the main advantages of home appliance networks. Below we present such services in our running example. Some of the services are taken from [8].

HVAC service (Heating): The HVAC service integrates the features of the air-conditioner, the two thermometers, and the ventilator. This service achieves energy-efficient air-conditioning of a room. If the room is cooler than the temperature set point, the HVAC service operates the air-conditioner in the heating mode. To efficiently warm the room up, the HVAC service turns the ventilator on to provide warmer outside air if the room temperature is cooler than the outside temperature. In this case the ventilator will keep operating until the room temperature reaches the outside temperature. If the room temperature is warmer than the temperature set point, the HVAC service operates the air-conditioner in the fan mode.

Air-cleaning service: The air-cleaning service uses the smoke sensor and the ventilator to automatically clean the air in the room. When smoke is detected, this service automatically turns on the ventilator. The ventilator keeps operating until the sensor senses no smoke. When the air is cleaned, the ventilator will be turned off.

Home theater service: The home theater service uses the TV set, the DVD player, the illuminometer, the lights and the curtain blinds. When activated, this service turns on the TV set and the DVD player. At the same time, the curtain blinds in the room are drawn down, and the lights are adjusted to the optimal level based on the current brightness of the illuminometer.

Energy saving service: This service aims to conserve energy consumption by turning off unnecessary appliances. For example, when the power of the TV set is off, it is useless to keep the power of the DVD player turned on. This service will turn off the DVD player in such a situation.

2.3 Feature Interactions of Services

In this section, we show several examples of interactions between the services shown in Section 2.2.

Example1: A feature interaction occurs between the HVAC service and the air-cleaning service. Consider the following situation: The room temperature is 15˚C, the outside temperature is 8˚C, and there is smoke in the room. The temperature set point of the HVAC service is 21˚C. Now suppose that the HVAC service is operating the air-conditioner in the heating mode. In this case, the HVAC service tries to turn off the ventilator to prevent cool outside air from flowing into the room. On the other hand, the air-cleaning service tries to turn on the ventilator to clean the room air.

Example2: This interaction occurs between the home theater service and the energy saving service. Consider the following scenario: Initially, the power of the TV set is OFF. The energy saving service checks and hence knows that the TV set is OFF and thus tries to turn off the DVD player. At the same time, the home theater service is activated and turns the TV set on. As a result, the TV set is turned on while the DVD player is turned off.

Example3: Suppose that the HVAC service is operating the air-conditioner in the heating mode to warm up the room and that the air-cleaning service is using the ventilator to clean the room air. If the room temperature is higher than the outside temperature, cool outside air is taken by the ventilator. This lowers the efficiency of the HVAC service.
3 Detection of Feature Interactions with SPIN

To use SPIN, the system to be verified needs to be described in the Promela language. In the Promela language, a system is defined as a collection of processes that communicate via buffered channels and global variables. Properties to be verified are represented as LTL formulas. When the Promela code and an LTL formula are given, the SPIN model checker automatically determines whether or not the given property holds. If the property does not hold, SPIN produces a counterexample. Using this counterexample, one can easily detect the cause of the property violation.

In Section 3.1, we show a method for describing home appliance networks and its users. Next, in Section 3.2, we show a classification of feature interactions based on their causes. We also devise LTL formulas to detect these feature interactions.

3.1 Describing Home Appliance Networks and Users in Promela

This section shows a method for describing home appliance networks and its users in Promela. A home appliance network is modeled as three components: the environment, appliances and services. The environment consists of several elements, such as the room temperature and smoke. The state of these elements is changed by the effects from appliances. The appliances are operated by services. User execute these services.

Figure 2 shows a model of home appliance networks and its users. In Figure 2, there are two users, two services (the HVAC service and the air-cleaning service), three appliances (an air-conditioner, a ventilator and a smoke sensor), and three elements of the environment (room temperature, smoke, and brightness). In this figure, the arrows represent relations between the four components. UserA executes the HVAC service. The HVAC service operates the air-conditioner and the ventilator. The air-conditioner has an effect on the room temperature, and the ventilator has effects on the room temperature and the smoke.

In this section, we present a method for describing these four components: the environment, appliances, services and users.

In Promela programs, the states of these components are represented by variables. The types of these variables are defined, for example, as follows:

```c
#define tTemp int
#define tPower int /* ON or OFF */
#define OFF 0
#define ON 1
```

*tTemp* is the type of the variables that represent temperatures. *tPower* is the type of the variables that represent the power of appliances. Variables of *tPower* type can take OFF or ON, which is internally represented as an integer 0 or 1.

3.1.1 The Environment

The elements of the environment are defined as global variables. A room temperature *temp_in* and an outside temperature *temp_out* can be defined as follows:

```c
tTemp temp_in; tTemp temp_out;
```

To model an unpredictable environment we let these variables take arbitrary values when they are read by appliances. For example, the room temperature changes even when the air-conditioner is not working. This modeling technique can be found in, for example, [2].

We represent the effects of the appliances on the environment by Boolean-valued formulas over the variables that represent the state of appliances and the state of the environment. As shown later, these formulas allow us to detect conflicting effects. For example, the effect *temp_in_up*, which indicates the presence of some appliance that is warming the room up, is defined as follows:

```c
#define temp_in_up
    ((AC_power == ON && AC_Mode == Heater)
    ||(ventilator_power == ON
        && temp_in < temp_out))
```

Here *AC_power*, *AC_mode* and *ventilator_power* respectively represent the power
of the air-conditioner, the mode of the air-conditioner, and the power of the ventilator. This Boolean formula evaluates to true when the air-conditioner is working in the heating mode or when the outside temperature is warmer than the room and the ventilator is working.

### 3.1.2 Appliances
Appliances are described with variables and macros. The variables represent the state of the appliances. The macros represent the methods of the appliances, such as setting the power to ON or setting the mode to a particular mode.

The state of the appliance is defined as global variables as was done with the environment. For example, the variables `power`, `AC_temp` and `AC_Mode`, which represent the power, the temperature set point and the mode of the air conditioner, are defined as follows:

\[
\begin{align*}
tPower & \quad AC_power = OFF; \\
tTemp & \quad AC_Temp = 25; \\
tMode & \quad AC_Mode = FAN;
\end{align*}
\]

The methods of the appliances are invoked by services. The methods can have arguments. The behavior of each method consists of reading the variables of the environment, writing/reading the variables of its own appliance and returning a value to the caller service.

### 3.1.3 Services
Each service is modeled by two Promela processes; one of which represents the behavior of the service and the other of which represents the communication between the service and users.

The process representing the behavior controls appliances by executing their methods. A method invocation is performed by executing the macro corresponding to the method. Special local variable `r_value` is used to store the return value from a method.

Figure 3 shows the Promela code of the process that describes the behavior of the HVAC service. At lines 2, 3, local variables are defined. At lines 4–13, the behavior of the HVAC service is described. At line 4, this service waits until the variable `HVACState` is set to `START`. Line 5 is a do-statement which is an iteration statement of Promela. At lines 6–10, several macros are executed.

The state of a service is represented by a set of global variables. The process that deals with communication updates the state in response to the reception of a message from a user. Figure 4 shows the process for the HVAC service.

For each service, global variable `ServiceState` is declared to control the start and stop of the service. This variable takes two values: `STOP`, `START`. The service waits until a user sets this variable to `START`, then performs its execution. When the variable is set to `STOP` by a user, the service stops and waits until a user sets this variable to `START` again. In addition to `ServiceState`, other global variables may be used. For example, the HVAC service has variable `HVACSetTemp` to represent the temperature set point.

For each such global variable, a message channel is declared. `MC_HVAC_State?HVAC_State` and `MC_HVAC_SetTemp?HVAC_SetTemp` are the message channels for `HVACState` and `HVACSetTemp`. To set the variable to a particular value, users send a message in the message channel. For example, `MC_HVAC_state?HVAC_state` takes a message in the message channel `MC_HVAC_state` and stores it in variable `HVAC_state`.

### 3.1.4 Users
Users control services by sending messages to message channels. For example, in the case of the HVAC
service, a user may send the value of the temperature set point, as well as a signal for start and stop, as shown in Figure 5. In this figure, UserA sets the variable SetTemp, a temperature set point, to 21°C and sets the variable HVAC_State to START. After the HVAC service starts, the user sets HVAC_State to STOP to stop the HVAC service.

3.2 Representing Correctness Claims as LTL Formulas

In this section, we classify feature interactions based on their causes. To detect feature interactions using SPIN, we need to represent the absence of each type of feature interactions as an LTL formula. Two kinds of temporal operators are used in this paper: always and eventually. The operator “always” is represented as $[]$. A formula $[] P$ evaluates to true if $P$ is always true in all system executions. The operator “eventually” is represented as $<>$. A formula $<> P$ evaluates to true if $P$ eventually becomes true in all system executions.

In general, feature interaction occurs when conflicting accesses are attempted to the same resource. Since there are four types of components (i.e., users, services, appliances and the environment), We have a simple classification as follows:

**Interaction with services:** Two users send conflicting commands to the same service.

**Interaction with appliances:** Two services are attempting conflicting operations on the same appliance.

**Interaction with the environment:** Two appliances have conflicting effects on the same element of the environment.

Figure 6 shows the examples of these interactions. In this figure, UserA and UserB can send conflicting commands to the HVAC service. The HVAC service and the air-cleaning service can operate the ventilator with conflicting purposes. The air-conditioner and the ventilator have conflicting effects on the room temperature.

3.2.1 Interactions with Services

Interactions occur with services when two users send conflicting commands to the same service. This type of interaction can be detected by checking if an incoming message conflicts with the previous message.

We consider a message $m$ from a user to a service to be conflicting if the following three conditions are met: the service has already received another message $m'$; $m'$ was issued by a different user; and the command of $m$ is different from that of $m'$.

To detect conflicting messages, we modify send/receive statements of user and service processes in two respects. First, the identity of sender users is attached to every message. Second, additional Promela code is inserted just after each receive statement of a service. For example, the receive statement of line 3 in Figure 4 will be modified as follows:

```plaintext
MC_HVAC_State?user,HVAC_State;
if ::(MC_HVAC_State?[_,_] && !MC_HVAC_State?[eval(user),_] && !MC_HVAC_State?[_ ,eval(HVAC_State)])
   -> HVAC_error = 1;
::else -> skip;
fi;
```

$MC?[a1,a2]$ evaluates to true iff in channel $MC$ a message $(a1,a2)$ exists. Underscore (_) is used as a wildcard. eval() is a function that returns the current value of the variable given as an argument.

The variable HVAC_error is used to record the occurrence of interactions. We let each Service have variable Service_error. If there is a conflicting message in the message channel, Service_error is set to 1. Accordingly, interactions with each Service can be detected by checking the following LTL formula:

$<> (Service_error == 1)$
#define AC_SetMode(mode) {
  AC_Power == ON;
  if\n    \:\:(MC_AC_Mode?[_] \&\&
    !(MC_AC_Mode?[mode]))
    \-> AC_error = 1;\n  \:\:else \-> skip;\n  fi;\n  MC_AC_Mode!mode;\n  MC_AC_Mode?AC_Mode;}

Figure 7. Method SetMode of The Air-conditioner

3.2.2 Interactions with Appliances

Application-level interactions occur when several services try to execute conflicting operations on the same appliance. This type of interaction occurs in two situations: A1) two services try to change the state of the appliance to different states and A2) one service reads the state of an appliance when another service is changing the state.

We define variable appliance_error for each appliance to represent whether or not a feature interaction has occurred (0: not occurred, 1: A1, 2: A2). appliance_error is updated when a feature interaction occurred.

**Detection of A1:** Interaction A1 occurs when two services try to set the same variable of an appliance to different values. To detect this interaction, we can use a similar way as interaction S1. We translate a variable assignment statement into a pair of a send statement and a receive statement, mimicking value assignment as message passing. When the send statement is executed, interaction A1 can be detected by checking whether or not a conflicting message has already been sent. For example, Figure 7 shows method AC_SetMode(mode) of the air-conditioner.

In Promela, if a message channel is full, the send statement waits until the message channel becomes non-full. For a method to receive the message sent by itself, the buffer size of message channel must be 1. This guarantees that, for example, when the receive statement is executed at line 10 in Figure 7, the only message in the message channel is the message sent at line 9. Hence, the message that can be received at line 10 is only the message sent at line 9.

**Detection of A2:** Interaction A2 occurs when one service tries to read a variable when another service is changing the value of the variable. As a result of this interaction, the value obtained by the former service becomes different from the actual value of the variable. This interaction can be detected by checking, whenever a method reads a variable, if the channel associated with that variable contains a new value different from the current one. For example, the macro TV_CheckPower, which is used to check the power of the TV set by services, is described as follows:

#define TV_CheckPower() {
  true;
  if\n    \:\:(MC_TV_Power?[_] \&\&
    !(MC_TV_Power?[eval(TV_Power)])
    \-> TV_error = 2;\n  \:\:else \-> skip;\n  fi;\n  r_value = TV_Power;}

For each appliance, interactions on it can be detected by checking if the value of appliance_error is 1 or 2. When checking interaction A1, for example, one can use the following LTL formula:

!<> AC_error == 1

3.2.3 Interactions with the Environment

Interactions with the environment occur when two appliances have conflicting effects on the same element of the environment. Hence this interaction occurs in the following two situations: E1) there are two different kinds of effects occurring on the same element of the environment simultaneously, and E2) some appliance reads the state of an environment element while some effect on that element is existing.

**Detection of E1:** This interaction can be detected by checking that continuous conflicting effects on the same environment element never occurs. This property is represented by an LTL formula as follows:

LTL: !<> [\] (e_eff1 \&\& e_eff2)

where e_effi is the formula which represents that some appliance has an effect effi on the element e. This LTL formula asserts that the two different effects on the same element never occurs simultaneously. When the number of the types of effects is more than 2, by checking all pairs of effects, one can detect this type of interactions. For example, when the number of types is 3, the LTL formula can be described as follows:
LTL: !<> ( [] (e_eff1 && e_eff2) || [] (e_eff1 && e_eff3) || [] (e_eff2 && e_eff3) )

Detection of E2: This interaction can be detected by checking if some effect is existing on an element of the environment when a service reads the state of the same environment element. For each environment element e, variable e_read is used to detect such a situation. We add the following statements to all methods that read the value of the element e at the point just before a statement that reads the value of the element.

```
e_read = 1; e_read = 0;
```

The value of e_read becomes 1 only if some method has just read the state of e. As a result this type of interaction can be detected by using the following LTL formula:

```
LTL: !<> (e_read && (e_eff1 || .. || e_effn))
```

(e_eff1 || .. || e_effn) represents that no effect is occurring on e. Hence, this LTL formula asserts that effects on the environment element e never exist while the state of e is being read.

4 Experiment

We conducted an experiment, in which we attempted to detect interactions caused by any pair of the four services of our running example (see Section 2.2). In this experiment, we assumed that there are two users and each user executes a single service. The experiment was conducted on a WindowsXP PC with a 900MHz PentiumIII and 512MB memory. SPIN was used with partial order reduction enabled.

In our running example, there are ten appliances as shown in Figure 1. Each appliance has one or two variables and two to six methods. The lines of code of the HVAC service, of the air-cleaning service and of the home theater service are all approximately 50 lines. The energy saving service is described in around ten lines.

Detection of service-level interactions was conducted by enforcing the users to run the same service. Unlike the other three services, the HVAC service requires the user to specify the temperature set point. In this experiment, this value is set to 21 °C or 25 °C.

For each of the two interaction types, A1 and A2, we check whether or not the interaction occurs for the ten appliances. Hence, we run a verification 20 times for each pair of services.

To detect interactions with the environment, we run a verification eight times, for there are two types of interactions, E1 and E2, and four environment elements.

4.1 Verification Results

The verification results are summarized in Table 1. This table shows the interactions detected between two services. S1, A1, A2, E1 and E2 indicate the type of interactions (services (S1), appliances (A1, A2), the environment (E1, E2)). Each symbol is followed by the component with which the interaction occurs. For example, when one user executes the HVAC service and the other user executes the air-cleaning service, a type A1 interaction with ventilator can occur. In the same case, types E1 and E2 interactions with room temperature and a type E2 interaction with smoke can occur.

Some performance figures are shown in Table 2 for the pair of the HVAC service and the air-cleaning services. Table 2(A) shows the results for interactions with appliances, while Table 2(B) shows those for interactions with the environment. These tables show the verification results (true indicates that an interaction was found), the execution time, and the number of states explored for each appliance or environment element. The results for the appliances that are not used by the two services are omitted. As shown in these tables, the time required for verification is fairly reasonable.

Using counterexamples provided by SPIN, we detected scenarios leading to interactions. Below we show several examples of such scenarios.

Type A1 interaction with the ventilator between the HVAC service and the air-cleaning service: The room temperature is warmer than the outside temperature (in winter), and there is smoke in the room. The HVAC service and the air-cleaning service are both running, and the HVAC service operates the air-conditioner in the heating mode to warm up the room. The HVAC service calls method SetPower(OFF) of the ventilator to turn it off, to prevent cool outside air from flowing into the room. On the other hand, the air-cleaning service executes method SetPower(ON) of the ventilator to turn it on to clean the room air. As a result, the conflicting operations of the ventilator are executed at the same time.
Type E1 interaction with the room temperature between the HVAC service and the air-cleaning service: This scenario is the same as the third example in Section 2.3.

Type E2 interaction with the room temperature between the HVAC service and the air-cleaning service: Suppose that the room temperature is cooler than the outside temperature (in summer) and that the air-cleaning service is operating the ventilator to clean the room air. In this situation, the ventilator warms the room up. Now the HVAC service operates the air-conditioner in the cooling mode, and executes the method that measures the current room temperature. Since the ventilator is having an effect on the room temperature, the HVAC service can erroneously recognize the room temperature as if it was higher than the actual value, resulting in execution of unnecessary cooling.

Type A1 interaction with the DVD player between the home theater service and the energy saving service: The power of the TV set is OFF at the beginning. The energy saving service checks the power of the TV set and knows that the power is OFF, thus tries to execute SetPower(ON) to turn off the DVD player as well. At this time, the home theater service is activated and tries to execute SetPower(OFF) to turn the DVD player on. As a result, the conflicting operations to DVD player are executed.

Type A2 interaction with the TV set between the home theater service and the energy saving service: This scenario is the same as the second example in Section 2.3.

5 Related Work

The feature interaction problem has long been recognized in the field of telephone communication systems [5]. For example, in [1], a method for detecting feature interactions with the SPIN model checker is proposed.

Unlike telecommunication systems, modeling home appliance networks requires us to consider the “physical” environment which might involve, for example, temperature or brightness. In the context of interaction detection for intelligent building control systems, Metzger and Webel proposed an approach to deal with such physical elements of the environment [11]. The idea of their approach is to detect different services that access the same environment element. Unlike ours, their approach does not consider how the services affect the environment. As a result it easily

---

**Table 1. Interactions detected**

<table>
<thead>
<tr>
<th>HVAC service</th>
<th>Air-cleaning service</th>
<th>Home theater service</th>
<th>Energy saving service</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1(HVAC service) E2(room temperature)</td>
<td>E1(ventilator) E2(room temperature)</td>
<td>E2(room temperature)</td>
<td>HVAC service</td>
</tr>
<tr>
<td>E1(room temperature)</td>
<td>E2(brightness)</td>
<td>E2(room temperature)</td>
<td></td>
</tr>
<tr>
<td>E1(Air-cleaning service) E2(brightness)</td>
<td>E2(room temperature)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 2. Verification Results between The HVAC Service and The Air-cleaning Service**

<table>
<thead>
<tr>
<th>appliance</th>
<th>type</th>
<th>result</th>
<th>time(s)</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Air-conditioner</td>
<td>A1</td>
<td>true</td>
<td>1.57 x 10^7</td>
<td>9.9 x 10^9</td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>true</td>
<td>1.45 x 10^7</td>
<td>9.9 x 10^9</td>
</tr>
<tr>
<td>Thermometer (inside)</td>
<td>A1</td>
<td>true</td>
<td>1.46 x 10^7</td>
<td>9.9 x 10^9</td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>true</td>
<td>1.45 x 10^7</td>
<td>9.9 x 10^9</td>
</tr>
<tr>
<td>Thermometer (outside)</td>
<td>A1</td>
<td>true</td>
<td>1.55 x 10^7</td>
<td>9.9 x 10^9</td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>true</td>
<td>1.48 x 10^7</td>
<td>9.9 x 10^9</td>
</tr>
<tr>
<td>Smoke sensor</td>
<td>A1</td>
<td>true</td>
<td>1.52 x 10^7</td>
<td>9.9 x 10^9</td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>true</td>
<td>1.45 x 10^7</td>
<td>9.9 x 10^9</td>
</tr>
<tr>
<td>Ventilator</td>
<td>A1</td>
<td>false</td>
<td>2.65 x 10^-4</td>
<td>1.1 x 10^-2</td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>true</td>
<td>1.49 x 10^7</td>
<td>9.9 x 10^9</td>
</tr>
</tbody>
</table>

---

**Table 2. Verification Results between The HVAC Service and The Air-cleaning Service**

<table>
<thead>
<tr>
<th>appliance</th>
<th>type</th>
<th>result</th>
<th>time(s)</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>room</td>
<td>E1</td>
<td>false</td>
<td>2.50 x 10^-4</td>
<td>8.7 x 10^3</td>
</tr>
<tr>
<td>temperature</td>
<td>E2</td>
<td>false</td>
<td>2.34 x 10^-1</td>
<td>2.0 x 10^9</td>
</tr>
<tr>
<td>outside</td>
<td>E1</td>
<td>true</td>
<td>1.41 x 10^7</td>
<td>9.9 x 10^9</td>
</tr>
<tr>
<td>temperature</td>
<td>E2</td>
<td>true</td>
<td>1.51 x 10^7</td>
<td>9.9 x 10^9</td>
</tr>
<tr>
<td>smoke</td>
<td>E1</td>
<td>true</td>
<td>1.46 x 10^4</td>
<td>9.9 x 10^9</td>
</tr>
<tr>
<td>brightness</td>
<td>E2</td>
<td>false</td>
<td>2.50 x 10^-4</td>
<td>2.6 x 10^9</td>
</tr>
<tr>
<td>brightness</td>
<td>E2</td>
<td>true</td>
<td>1.48 x 10^4</td>
<td>9.9 x 10^9</td>
</tr>
</tbody>
</table>
Research that addressed the feature interaction problem of home appliance networks includes [8, 13]. In [8], a runtime detection method and a priority-based resolution are proposed. Our approach works at a higher abstract level than [8] in the sense that interactions detected by our approach might be resolved by prioritizing services. Thus, even when such runtime resolution exists, the results obtained through our approach can be used to identify the situations where the mechanism comes into play, resulting in a better understanding of the system behavior.

In [13], a static method for detecting feature interactions is proposed. However, this method is a conservative approximation method and thus can detect false feature interactions which will never occur in actual runs.

In [9], a method for verifying the behavior of services with the SMV model checker [10] is proposed. This method can also be used for feature interaction detection, it was not the main objective of [9], though. The work presented in this paper improves [9] in several ways: First, we classified interactions and devised the LTL correctness claim for each category. These LTL formulas allow systematic interaction detection, while in [9] correctness claims were constructed in an ad hoc manner. Another improvement came from our adoption of the Promela language. Unlike the SMV input language, Promela is similar to conventional procedural programming languages. This makes it much easier to describe the specification of appliances and services.

6 Conclusion

In this paper, we proposed a method for detecting feature interactions in home appliance networks. The proposed method uses SPIN, and LTL model checker. We classified interactions into several types and devised LTL formulas that represent the absence of these interactions. To demonstrate the usefulness of the proposed approach, we conducted an experiment. We checked whether or not feature interactions occur in our running example and successfully detected several interactions. By using counterexamples produced by SPIN, we also succeeded in obtaining scenarios leading to the interactions.

References