AUTOMATICALLY VERIFYING INTEGRATED SERVICES IN HOME NETWORK SYSTEMS

T. Matsuo[1]*, P. Leelaprute*[2]*, T. Tsuchiya*[1]*, T. Kikuno*[1]*, M. Nakamura*[3]*, H. Igaki*[3]*, and K. Matsumoto*[3]

[1]Graduate School of Information Science and Technology, Osaka University, Japan
{t-matuo, t-tutiya, kikuno}@ist.osaka-u.ac.jp
[2]Department of Computer Engineering, Faculty of Engineering, Kasetsart University, Thailand
pattara@cc.cpe.ku.ac.th
[3]Graduate School of Information Science, Nara Institute of Science and Technology, Japan
{masa-n, hiro-iga, matumoto}@is.naist.jp

ABSTRACT

This paper addresses the problem of verifying integrated services of home network systems which consist of one or more home appliances. Specifically, an approach is proposed that uses the SPIN model checker to verify integrated services automatically and exhaustively. The proposed approach encompasses a method for automatically translating integrated service descriptions into the Promela language, the input language of SPIN. Once a Promela code for a service description has been obtained, one can verify the integrated service against the properties represented as LTL formulae. The usefulness of the proposed approach is demonstrated through a case study using an example of a practical integrated service. The findings and experiences obtained in this case study offer some clues to adequately representing properties of interest for integrated services.

1. INTRODUCTION

A system that consists of home appliances connected via a LAN is called a Home Network System (HNS). This paper addresses the problem of verifying the behavior of services provided by HNSs.

Integrating features of different appliances can widely extend the capabilities of HNSs [6]. For instance, consider the following example of services provided by an HNS presented in [4]. This system consists of the following networked appliances: an air-conditioner, a ventilator, a window, two thermometers, and a smoke sensor. The air-conditioner operates in either of the two modes: the cooling mode and the standby mode.

HVAC Service [3]: HVAC achieves energy-saving air-conditioning of a room based on the set temperature. If the room is warmer than the set temperature, the HVAC service operates the air-conditioner in the cooling mode. To efficiently cool down the room, if the room temperature is warmer than outside, the ventilator is also turned on to provide fresh outside air. In this case the ventilator will be kept working until the room temperature reaches the outside temperature. If the room temperature is below the set temperature, on the other hand, HVAC has the air-conditioner operate in the standby mode.

Air-Cleaning Service: The Air-Cleaning Service, which involves the smoke sensor, is used to clean the air in the room. When the smoke sensor detects smoke, the service automatically opens the window and turns on the ventilator. When the sensor detects no smoke, the window and the ventilator are shut down.

How to integrate different appliances to accomplish the integrated services has not been widely studied. This lack of research may hinder development of reliable and high-quality services. To solve this problem, this paper presents a method of verifying HNS integrated services [5] with the SPIN model checker [2]. This method first translates HNS integrated services description into the Promela language, the input language of SPIN. With the Promela code and a property to be verified that is stated as an LTL formula, 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.

2. HNS DESCRIPTION LANGUAGE

In this section, we present a language for representing integrated services, proposed in [4]. An integrated service description has two parts: (a) a system description for the HNS and (b) a service description for the integrated services. In this service description language, both appliances and the environment that the appliances interact with are modeled as objects. It is assumed that the platform that executes the integrated service can read the values of the properties of any appliances.

ISBN 974-94418-9-3 ©2006 ECTI

The 21st International Technical Conference on Circuits/Systems, Computers and Communications

2.1. System Description

An HNS is described in the following format. A sentence that begins with # is a comment. Capital words (like SYSTEM, TYPEDEF, etc.) denote keywords.

```
SYSTEM HNS_name {
  TYPEDEF # Type definition
    type_name1 type1;
    :
  # Environment definition
  ENVIRONMENT env_name {
    PROPERTY
    # Environment properties declaration
      type_name1 env_property1 [:= init];
      :
  }
  # Appliance definition
  APPLIANCE appliance_name1 {
    PROPERTY
      # Appliance properties declaration
      type_name1 app_property1 [:= init];
      :
    METHOD # Method definition
      return_type1 method_name1([type_name
formal_para]*){
        PRE pre_condition;
        POST post_condition;
        ENV_R env_name.property_name;
        ENV_W env_name.property_name;
        RETURN return_value;
      }
      return_type2 method_name2(...){...}
      :
  }
  APPLIANCE appliance_name2 {...}
  :
}
```

The TYPEDEF section declares types used in the system. The language supports three types: Boolean, integer, and enumeration. An integer type is specified by the upper and lower bounds, e.g., {1..5}. An enumeration type is defined by enumerating its elements, e.g., {ON,OFF}.

The ENVIRONMENT section defines an environment object. The environment consists of only environment properties.

All appliances deployed in the HNS are declared in multiple APPLIANCE sections, each of which defines an appliance object. An APPLIANCE block comprises of the definitions of properties and the methods of the appliance.

A method is associated with a name and its return type. The return type must be a type that is declared in the TYPEDEF or void. The body of the method contains the pre/post conditions, the environment properties read/written, and the return value. If the pre condition of a method is $true$, the method can be executed. The post condition must be satisfied where the method returns. It is assumed that during the execution of a method, no other methods of the same appliance can be executed.

2.2. Service Description

The service description is in the following format:

```
DEPLOYED_SYSTEM HNS_name;
SERVICE
  service_name1([type_name formal_para]*){
  # Local variable declaration
  VAR type_name local_var1 [:=init];
      :
  # Appliance declaration
  APPLIANCE appliance1, appliance2,..;
  CONTENT # service content
    statement1;
    statement2;
    :
  }
  :
}
```

The keyword DEPLOYED_SYSTEM is used to specify the name of the HNS where the services are deployed. It imports a system description designated by the name.

A service can use local variables declared in the VAR subsection. Allowable types are either Boolean, integer, or enumeration. The APPLIANCE subsection designates appliance objects used in the service.

The CONTENT subsection describes the body of the service, which consists of one or more statements. Basically, the statements are sequentially executed one-by-one from top to bottom, as in ordinary procedural programming languages. Meta functions, end() and exit(), model abstract events. An end() returns $true$ when the user terminates the service. An exit() models a system call that is used by system to terminate the service.

3. MODEL CHECKING SERVICES WITH SPIN

Model checking is the process of exploring a finite state space to determine whether or not a given property holds. To use SPIN, the system needs to be described in the Promela language which is based on C.

SPIN provides a way of defining properties by linear temporal logic (LTL) [7] formulae defined separately from the code. An LTL formula states properties about the execution traces of a Promela program, where a trace is a sequence of states. The model checker determines whether or not all traces starting with the initial state satisfies a given LTL formula.

The following of this section describes how to translate system and service descriptions into the Promela language. To translate the HNS and services description into the Promela language, we make the following assumptions: i) the system and service descriptions have no syntax and semantic errors; ii) an environment property value can take an arbitrary value when a method reads/writes its value; iii) multiple services can be concurrently executed in an interleaving manner.

We implemented a C program that translates service descriptions into the Promela language. Below is the outline of the translation method.

3.1. System Description

We only allow the integer type as a primitive type. Enumeration types are handled by assigning a unique integer number to each element. For instance, we translate tPower {ON,OFF} as follows:

```
#define tPower int
#define ON 0
#define OFF 1
```

This does not prevent `tPower` from taking an incorrect value. For instance, a variable of `tPower` type can be set to 2. However, because of the assumption that the system and service descriptions have no errors, such a case never occurs.

Environment properties are represented as global variables as follows:

```
type_name1 env_property1 [:= init];
type_name2 env_property2 [:= init];
:
```

`env` is the name of the environment.

Like environment properties, appliance properties are defined as global variables.

An appliance method definition is translated into an inline definition as follows:

```
inline appname_methodname(argument){
  atomic{appname_semaphore == 1;
         appname_semaphore = 0;}
  pre_condition;
  (set_post_condition_true);
  [set_env_property;]
  [rvalue = return_value;]
  appname_semaphore = 1;
}
```

We use a semaphore to avoid the case where two methods of the same appliance are executed at the same time. The `pre_condition` is blocked until the pre condition becomes *true*. The `(set_post_ condition_true)` command is used to have the post condition hold.

The `set_env_property` command resets environment properties to arbitrary values if the method reads or writes the environment properties.

When the Promela program is compiled, each point of invocation of an inline is replaced with the text of the inline body. If the method has a return value, the return value is returned by storing it at a local variable, `rvalue`, of the caller process.

3.2. Service description

A single service is represented as a single process as follows:

```
proctype service_name(argument){body; END:skip}
```

The process has at its end an `END:` label and a `skip` statement which is a no-operation statement. This pair is used when the service is terminated by the system.

Local variables of a service are declared as local variables of a process. Special local variables, `rvalue` and `end`, are declared for each service. `rvalue` stores a method's return value. When `end` is set to 1, it means that the service is terminated by the user.

The appliance declaration is ignored because our assumption that the system and service descriptions have no errors ensures that no undeclared appliances are used.

A service content can be described in almost the same format as a service description. The statements are translated into statements that can be used in Promela. An invocation of a method is replaced by an `inline`. An `exit()` is translated into `goto END`.

4. CASE STUDY

We verify the HVAC and Air-Cleaning services described in Section 1 against the following properties.

- If the service is started, then the air-conditioner will be eventually turned on (P1).
- If the room temperature exceeds the set temperature, then the mode of the air-conditioner will be set to the cooling mode (P2). On the other hand, if the temperature is below the set temperature, then the mode of the air-conditioner will be set to the standby mode (P3).
- Once the ventilator is turned on, it will be kept operating until the inside temperature reaches the outside temperature (P4).
- Whenever the sensor device senses smoke, the ventilator will be turned on (P5) and the window will be opened (P6).
- If the window is opened, then the window will be kept open until the sensor no longer senses smoke (P7). Similarly, once the ventilator is turned on, it will be kept working until no smoke is detected (P8).

For example, P3, P5, P7, P8 are translated into following LTL formulae:

```
P3:[](!(inside_temp>set_temp)
      -><>(AirConditioner_mode==STANDBY))
P5:[](SmokeSensor_currentSmoke==1
      -><>(Ventilator_power==ON))
P7:[](Window_status==OPEN
      ->(Window_status==OPEN
         U SmokeSensor_currentSmoke==0))
P8:<>(Ventilator_power == ON
      ->(Ventilator_power == ON
         U SmokeSensor_currentSmoke == 0))
```

4.1. Verification Results

We used the SPIN model checker to verify the properties from P1 through P8, by running the two services in parallel. We used a WindowsXP PC with PentiumIII 900MHz and 512MB memory. The results of the verification are shown in Table 1, showing that the properties P3, P4, P5, P7, and P8 can be invalidated in concurrent service execution.

The counterexample of P8 showed that the ventilator can be turned off even though there is still smoke. This

is because the HVAC service turns off the ventilator if the room temperature is below the outside temperature. The counterexample of P4 is similar to P8. The violations of P4 and P8 are due to a functional conflict among the services, which is well known as a feature interaction [1].

We thought that the causes of the violation of P3, P5, and P7 did not stem from undesirable behaviors of the services. For example, the violation of P5 occurs if after environment property `Smoke` has been set to 1, `Smoke` is set to 0 before the ventilator is turned on. In this situation, the ventilator will never be turned on unless `Smoke` takes 1 again.

This finding suggests that nondeterministic behaviors of the environment should be carefully taken into account in describing temporal properties. Based on this observation, we modified the properties P3, P5, and P7 as follows.

```
P3':[](!(inside_temp>set_temp)
       ->(<>((AirConditioner_mode==STANDBY)
             || (inside_temp>set_temp)))
P5':[](SmokeSensor_currentSmoke==1
       ->(<>((Ventilator_power==ON)
             ||!(SmokeSensor_currentSmoke==1)))
P7':[](Window_status==OPEN
       ->((<>(SmokeSensor_currentSmoke==0)
          ->(Window_status==OPEN
             U SmokeSensor_currentSmoke==0))
         ||!(<>(SmokeSensor_currentSmoke==0)
          ->[](Window_status==OPEN))))
```

We verified the properties P3', P5' and P7'. The results of the verification are shown in Table 1.

The counterexample of P3' showed that the trace exists where the air-conditioner will never be set to the standby mode even if the room temperature remains below the set temperature. Once the ventilator has been turned on, the HVAC service checks only whether the inside temperature reaches the outside temperature. When the outside temperature is below the set temperature, the HVAC service does not set the air-conditioner to the standby mode even if the inside temperature is cooler than the set temperature.

In this example, the desirable properties for the services are classified into two types. One states that if the environment reaches a certain state, a method of an appliance is executed. The other states that if a method of an appliance is executed, the appliance will keep operating in a specific state, until the environment becomes a certain state. These properties are represented by LTL formulae as follows:

```
[]( p -> <>( q ) )
[]( p -> ( p U q ) )
```

However, these properties do not correctly represent desirable properties because we assume that the environment properties values change nondeterministically. Hence these LTL formulae should be modified as follow:

```
[]( p -> <>( q || !( P ) ) )
[]( p -> ( ( <>q -> ( p U q ) )
      || (!( <>q ) -> []p ) ) )
```

With these LTL formulae, desirable properties can be represented more appropriately. We think that these LTL specifications are useful to verify other HNS integrated services.

Table 1: Verification Results

property	result	time(sec)	states
P1	true	16	1.1×10^6
P2	true	44	2.8×10^6
P3	false	13	8.9×10^5
P4	false	1	2.9×10^1
P5	false	84	4.9×10^6
P6	true	52	3.2×10^6
P7	false	6	4.3×10^5
P8	false	1	5.2×10^1
P3'	false	27	1.9×10^6
P5'	true	91	5.7×10^6
P7'	true	46	3.1×10^6

5. CONCLUSION

This paper addressed the problem of verifying integrated services of home network systems. We proposed an approach that uses the SPIN model checker to verify integrated services automatically and exhaustively. The proposed approach encompass a method for translating integrated services into the Promela language. As a case study, we verified two HNS integrated services presented in [4] with the proposed approach. As a result of a verification, we found that one of the two services had an undesirable behavior, and that there were two problems when the services were run in parallel. This result shows the usefulness of the proposed approach. Furthermore, the findings and experiences obtained in this case study offered some clues to adequately representing properties of interest for integrated services of HNSs as LTL formulae.

6. REFERENCES

[1] Feature interaction in telecommunications and software systems vol. I-VIII. IOS Press, 1992-2005.

[2] G. J. Holzmann. The model checker SPIN. *IEEE Trans. Softw. Eng.*, 23(5):279–295, 1997.

[3] M. Kolberg, E. H. Magill, and M. Wilson. Compatibility issues between services supporting networked appliances. *IEEE Communications Magazine*, 41(11):136–147, 2003.

[4] P. Leelaprute, M. Nakamura, T. Tsuchiya, K. Matsumoto, and T. Kikuno. Describing and verifying integrated services of home network systems. In *The 10th Asia-Pacific Software Engineering Conference (APSEC2005)*, pages 549–558, December 2005.

[5] M. Nakamura, H. Igaki, and K. Matsumoto. Feature interactions in integrated services of networked home appliance. In *Proc. of Int'l. Conf. on Feature Interactions in Telecommunication Networks and Distributed Systems (ICFI'05)*, pages 236–251, June 2005.

[6] OSGi Appliance. The osgi service platform. `http://osgi.org`.

[7] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE symposium on foundation of computer science*, pages 46–57. IEEE Computer Society Press, 1977.