NAIST-IS-DD0261003

Doctoral Dissertation

Developing Integrated Services of Networked Home Appliances

– Implementation framework with Service Oriented Architecture and Feature Interaction Analysis –

Hiroshi Igaki

March 24, 2005

Department of Information Systems
Graduate School of Information Science
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Hiroshi Igaki

Thesis Committee:

Professor Ken-ichi Matsumoto	(Supervisor)
Professor Masatsugu Kidode	(Co-supervisor)
Associate Professor Hajimu Iida	(Co-supervisor)

Developing Integrated Services of Networked Home Appliances

– Implementation framework with Service Oriented Architecture and Feature Interaction Analysis – *

Hiroshi Igaki

Abstract

Recent advancements in processors and networks has allowed emerging technologies to *network* various home electric appliances, including TVs, air-conditioners, lights, DVD players, and refrigerators. A system consisting of such networked home appliances is generally called a *Home Network System (HNS)*. 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 a more comfortable and convenient living environment for the users. HNS is considered to be one of the next-generation value-added services in a ubiquitous computing environment.

This dissertation presents two main contributions to the effective development of the HNS integrated services.

*Doctoral Dissertation, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD0261003, March 24, 2005.

Design and Implementation with Service Oriented Architecture

First, we propose a novel framework to design and implement the HNS integrated services. The proposed framework is characterized by the extensive use of a *Service Oriented Architecture (SOA)*.

The conventional approach to implement the HNS integrated services adopts the *Server Centralized Architecture (SCA)*, where an intelligent *home server* controls all the networked appliances in a centralized manner. To achieve each integrated service, the home server sends control commands to the appliances in a certain order. Thus, the mechanism of SCA-based HNS is quite intuitive. However, due to its centralized nature, the SCA-based HNS suffers from a concentration of load as well as a decline in reliability, interoperability, and system extendibility.

To cope with this problem, we use the SOA for implementation of the HNS integrated services. In the proposed framework, the appliances export their own features as services. By introducing a new concept — *a service layer* on top of the proprietary interface of each appliance, the appliance can autonomously execute the services exported by any other appliance with a standard procedure. Thus, the appliances are loosely coupled via the exported services. This enables more flexible, balanced and reliable HNS integrated services. We present a framework to design and implement the integrated services based on SOA, and then illustrate a prototype system developed with *Web services*.

We also propose a graph-based method to evaluate the HNS from the viewpoints of reliability, workload and coupling. With the proposed evaluation metrics, we conduct a comparative evaluation between the proposed and the conventional systems.

Feature Interaction Analysis in HNS Integrated Services

The second contribution of the dissertation is to address the problem of *feature interactions* in the HNS integrated services. The feature interaction generally refers to inconsistent conflict between multiple services, which is never expected from single services' behaviors. This problem has been originally studied within the area of telecommunication services.

The feature interaction problem occurs in the HNS integrated services as well, since multiple home users may activate different integrated services, simultaneously. The feature interaction decreases the total quality of services, and even leads to the system shutting down. Therefore, efficient detection and resolution of feature interactions are indispensable in guaranteeing the safe and comfortable living environment of home users. However, the feature interactions in this emerging domain are not yet well studied.

Our goal is to formulate the feature interaction problem within the HNS integrated services and propose an efficient analysis method for it. In the formulation, we first model each appliance as an *object* consisting of properties and methods, where a property represents an internal state of the appliance, and a method abstracts a feature. Each method consists of a pre-condition and a post-condition, and refers or updates values of some properties. Similarly, we also construct a model for *the home environment*.

Within the model, we formalize two types of feature interactions: *device interaction* and *environment interaction*. The device interaction is in *direct* conflict among features of the same appliance device. This conflict arises when multiple HNS integrated services simultaneously trigger methods that update an appliance property in a different way. On the other hand, the interaction environment is in *indirect* conflict among different appliances via the HNS environment. This conflict arises when multiple HNS integrated services simultaneously trigger different appliances so that they

try to perform an inconsistent update of an environment property.

Based on a formulation, we implement a feature interaction detection system, and conduct a case study of interaction detection for practical HNS integrated services. The proposed framework is quite generic and is well applicable to any HNS, including both SCA-based and SOA-based systems. We also discuss several resolution schemes for the detected interactions.

Overview of Dissertation

This dissertation is organized as follows. In Chapter 1, we summarize the background and related topics, and describe an outline of the dissertation. In Chapter 2, we present preliminaries for the HNS integrated services with practical example scenarios. In Chapter 3, we propose the SOA-based framework for the design, implementation and evaluation of the HNS integrated services. Chapter 4 discusses the feature interaction problem within the domain of the HNS integrated services. Finally, in Chapter 5, we conclude this dissertation with a summary and future work.

Keywords:

Integrated Services, Service Oriented Architecture, Feature Interaction and Home Network

List of Major Publications

Journal papers (Referred)

[1-1] Hiroshi Igaki, Masahide Nakamura, Haruaki Tamada, Ken-ichi Matsumoto, “Implementing Integrated Services of Networked Home Appliances Using Service-Oriented Architecture”, *Journal of Information Processing Society of Japan*, Vol.46, No.2, pp.314-326, Feb. 2005 (in Japanese).

International Conference Papers (Referred)

[2-1] Masahide Nakamura, Hiroshi Igaki, Ken-ichi Matsumoto, “Feature Interactions in Integrated Services of Networked Home Appliances -An Object-Oriented Approach-”, *Proc. of 8th International Conference on Feature Interactions in Telecommunications and Software Systems(ICFI2005)*, Jun. 2005(to Appear).

[2-2] Masahide Nakamura, Hiroshi Igaki, Haruaki Tamada and Ken-ichi 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.

[2-3] Hiroshi Igaki, Masahide Nakamura and Ken-ichi Matsumoto, “Design and evaluation of the home network system using the service oriented architecture”, *Proc. of International Conference on E-business and Telecommunication Networks (ICETE2004)*, Vol.1, pp.62-69, Aug. 2004.

Domestic Workshops and Symposiums

[3-1] Hiroshi Igaki, Masahide Nakamura, Ken-ichi Ishii, Youhei Kushido, Ken-ichi Matsumoto, “Runtime Feature Interaction Detection and Resolution in Integrated

Services of Networked Home Appliances", *Technical Report of IEICE.*, Vol.IN2004-320, pp.373-378, Mar. 2005 (in Japanese).

[3-2] Hiroshi Igaki, Masahide Nakamura, Ken-ichi Matsumoto, "Detecting Feature Interactions in Integrated Services of Networked Home Appliances" , *Technical Report of IEICE.*, DE2004-108, pp.11-16, Oct. 2004 (in Japanese).

[3-3] Hiroshi Igaki, Haruaki Tamada, Masahide Nakamura, Ken-ichi Matsumoto, "A Design and Evaluation Metrics of the Home Network Systems Using the Service Oriented Architecture" , *Technical Report of IEICE.*, IN2003-314, pp.333-338, Mar. 2004 (in Japanese).

Acknowledgements

During the course of this work, I have been fortunate to have received assistance from many individuals.

First, I would like to thank my supervisor Professor Ken-ichi Matsumoto. The many opportunities he gave me to proceed my research have stimulated many of my interests and allowed me to gain more experience in my research field.

I am also very grateful to the members of my thesis review committee: Professor Masatsugu Kidode and Associate Professor Hajimu Iida for their invaluable comments and helpful criticisms of this thesis.

I would likely to deeply thank Assistant Professor Masahide Nakamura for his support, patience and encouragement throughout my graduate studies. He led me to the research domain of Service Oriented Architecture and Feature Interaction. I have learned quite a lot from his extensive knowledge in my research domain and many brilliant and creative ideas. His technical and editorial advice was essential to the completion of this dissertation and has taught me innumerable lessons and insights on the workings of academic research in general.

I thank also many collegues in the Graduate School of Information Science, NAIST, who gave me many useful comments. Especially, I wish to thank Haruaki Tamada, a member of the Web Services Research Group, who gave me invaluable assistance with the experiments and with implementing our programs.

The other members of my Research Group likewise provided important advice and the encouragement to expand existing boundaries about my research domain. They included Ken-ichi Ishii, Youhei Kushido, Hiroki Yamauchi and Takahiro Kimura. Thanks also to all members at the Software Engineering Lab. for providing a confortable working atmosphere.

The research in this dissertation has been supported by Grant-in-Aid for 21st century COE Research(NAIST-IS Ubiquitous Networked Media Computing).

Abbrevations

A:
AXIS—Apache eXtensible Interaction System
API—Application Programming Interface

B:
BPEL4WS—Business Process Execution Language for Web Services

C:
CEBus—Consumer Electronic Bus

D:
DMI—Device Method Invocation

E:
ECHONET—Energy Conservation and HOmecare NETwork

F:
FSIG—Full Service Integration Graph

H:
HAVi—Home Audio/Video Interoperability
HNS—Home Network System
HNS-SCA—Home Network System based on Server Centralized Architecture
HNS-SOA—Home Network System based on Service Oriented Architecture

I:
IEEE1394—Institute of Electrical and Electronics Engineers 1394

M:
MAI—Multiple Action Interaction

S:
SCA—Server Centralized Architecture
SDK—Software Development Kit
SDP—Sum of Disjoint Products
SIG—Service Integration Graph

SMI—Service Method Invocation
SOA—Service Oriented Architecture
STI—Shared Trigger Interaction

U:
UDDI—Universal Description, Discovery, and Integration
UML—Unified Modeling Language
UPnP—Universal Plug and Play
URI—Uniform Resource Identifier

W:
W3C—WWW Consortium WS-CDL—Web Services Choreography Description Language WSDL—Web Services Description Language

X:
XML—eXtensible Markup Language

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1. Background

The Home network system (HNS) is an emerging trend in home appliances that features built-in communication capabilities, providing us with added convenience. This HNS provides many applications and services for home users, for instance, remote control (outside home), group control, monitoring, etc [12][42][28]. One of the major HNS applications is the *integrated service of networked home appliances* (called simply *HNS integrated service* in the following). The HNS integrated service orchestrates different home appliances via a network in order to provide a more comfortable and convenient living environment for users.

A typical HNS integrated service includes:

Coming Home Service: When a user comes home, lights and an air-conditioner are turned on with appropriate illumination and temperature.

DVD Theater Service: When a user turns on a DVD player, lights become dark, and 5.1ch speakers are selected while the volume is automatically adjusted [35].

Problems in realizing these integrated services exist. In the following Sections 1.1.1 and 1.1.2, we present two problems of conventional HNS applications.

1.1.1 Conventional HNS architecture

The conventional approach to implementing the integrated service adopts the *Server Centralized Architecture (SCA)*, where a sophisticated server (called *a home server*) plays the role of a conductor. The home server controls all the networked appliances in a centralized manner, by sending control commands to the appliances in a certain order [19, 31, 35]. Since the server undertakes all the intelligent tasks of the orchestration, the structure of SCA is quite simple and intuitive.

However, as networked appliances get more sophisticated and diversified, the conventional SCA will suffer from the following problems:

Reliability, Load Concentration: Since all appliances are controlled by a centralized server, a crash of the server makes all the integrated services unavailable. Also, the number of connected appliances directly reflects the heavy workload of the server.

System Extension: Features of appliances that are not compatible with the server cannot be used in the integrated services. This limitation will become an obstacle to system extension for future appliances.

Interoperability: Since the home server needs to know the underlying protocols of all networked appliances, implementation of the server middle-ware becomes complex. Also, the server and the appliances are tightly coupled. Hence, guaranteeing interoperability between appliances is difficult, especially when the versions of the protocols and the appliances are updated.

The first goal of this dissertation is to address the new design of the HNS integrated service, and its evaluation through practical integrated services.

1.1.2 Feature Interaction Problem

Various home appliances become parts of HNS integrated services (TVs, DVD players, Phones, Lights, Air-Conditioners, etc). As a result, many new services are being developed and deployed in order to achieve the various requirements of home customers.

When such integrated services are executed simultaneously, functional conflicts can occur between the functions of each appliance. This conflict is recognized as feature interaction, and it becomes a serious obstacle preventing service execution.

The HNS environment has the following features regarding feature interactions.

Diversity: The appliance used in HNS differs for every user. The relation among appliances also differs for every user. Therefore, the dependency with which feature interactions occur beforehand is not known.

Changeability: The HNS environment is easily changed by an additional purchase and a change of appliances. A flexible feature interaction detecting method with the ability to follow change of HNS environment is required.

Multiple Users: When such integrated services are executed simultaneously, functional conflicts may occur between the functions of each appliance.

Appliance Interactions and Environment Interactions: The HNS environment will almost always be embedded in a physical environment, which provides responses to a stimuli and which is not part of the actual software system. Because of this specific role of the environment, the identification and resolution of interactions that occur within the system are not enough. Interrelationships that are introduced by interactions with the environment must also be considered [32].

The second goal of this dissertation is to formulate feature interaction problems among HNS integrated services.

1.2. Main Results

1.2.1 Implementation Framework using Service Oriented Architecture

This dissertation presents an alternative architecture for HNS. Specifically, we propose to apply the service oriented architecture (SOA, for short) [18] to HNS. Basically, SOA is architecture to integrate distributed self-contained services using loose coupling and well-defined interfaces. We assume next-generation home electric appliances are intelligent enough to process service transactions with their own processors and network devices.

In the proposed method, each appliance is divided into two layers: a *service layer* and a *device layer*. Our key idea is to export features of each appliance as methods of the service layer, and to make the features directly available from other appliances in an open and standard manner (i.e., SOAP/XML). Thus, the appliances can autonomously collaborate with each other to build the HNS integrated services. Since the proposed SOA-based HNS(HNS-SOA) does not require a centralized server, it is expected to be more scalable and fault-tolerant. Also, more sophisticated and flexible integrated services can be developed.

In addition, we conducted a practical implementation and a quantitative evaluation of the proposed architecture. We implemented practical HNS integrated services using the Java Web Service, and we evaluated our framework and conventional HNS from the following viewpoints.

Reliability: the probability that the integrated services are operational in the HNS.

Workload: the workload of each component (service or centralized server) imposed when performing integrated services in HNS.

Coupling: the degree of dependence of a component against other components.

The result shows that HNS-SOA is superior to SCA-based HNS(HNS-SCA) comparatively in every viewpoint. Therefore, we show how the HNS-SOA framework is well applicable to practical HNS integrated services.

1.2.2 Feature Interaction Analysis

We formulate the feature interaction problem within the HNS integrated services, and analyze it efficiently. In the formulation, we first model each appliance as an *object* consisting of properties and methods, where a property represents an internal state of the appliance, and a method abstracts a feature. Each method consists of a pre-condition and a post-condition, and refers or updates values of some properties. Similarly, we also construct a model for *a home environment*.

Within the model, we formalize two types of feature interactions: *device interaction* and *environment interaction*. The device interaction is in *direct* conflict among features of the same appliance device. This conflict arises when multiple integrated services simultaneously trigger some methods that update an appliance property in a different way. On the other hand, the environment interaction is in *indirect* conflict among different appliances via the HNS environment. This conflict arises when multiple integrated services simultaneously trigger different appliances so that these appliances try to perform an inconsistent update of an environment property.

Based on the formulation, we implement a feature interaction detection system, and conduct a case study of interaction detection for practical HNS integrated services. The proposed framework is quite generic and is well applicable to any HNS, including both

HNS-SCA and HNS-SOA systems. We also discuss several resolution schemes for the detected interactions.

1.3. Overview of the Dissertation

This dissertation is organized as follows: In Chapter 2, we describe preliminaries for the HNS integrated services with practical examples. In Chapter 3, we propose the HNS-SOA application. We give a HNS-SOA design as the Service Integration Graph after explanation of the key idea. Next, we present an implementation template for the HNS integrated services with Java Web Service. By means of graph-based algorithm SDP (Sum of Disjoint Products), we evaluate both HNS-SOA and conventional HNS.

In Chapter 4, we propose a new feature interaction detection algorithm. We also formulate the feature interaction problem in the integrated services of the home network system. Specifically, we define two types of interactions: device interactions and environment interactions. We conduct a case study of interaction detection to demonstrate the effectiveness of the proposed method.

Finally, in Chapter 5, we conclude this disstertation with a summary and future works.

Chapter 2

Preliminaries

2.1. A Home Network System and its Applications

Computer networks have existed for more than thirty years, but only in the last several years have they become popular in homes. Today, many millions of households all over the world have adopted home networking.

Moreover, the emergence of high-speed, multi-layer, in-home networks will integrate traditional home automation and control technologies (such as X10 [55] and CEBus [8]) with media rich applications such as voice and video conferencing. As for the devices connected to a home network, not only the PC but also various appliances such as TVs and Air-Conditioners, are connected by LAN or wireless. Using such a HNS environment, applications as shown below are developed and proposed.

Health Monitoring System: Remote monitoring systems of the health status of elderly at home are developed [36, 46, 9]. In this research domain, wearable health sensors have been developed and embedded within a form factor of, for example, a ring [41] or a wristwatch [30]. By combining these wearable sensors with measurement devices embedded in home surroundings, advanced multiparamet-

ric health monitoring may be achieved [14, 26, 27, 39, 43].

Remote Control of Appliances: A home automation system like heating and cooling etc., can be turned on by telephone or other network technology[3, 38, 21].

Integrated Services with Appliances: Occupancy sensor technologies save energy and money by limiting lighting, appliances, and heating and cooling use when rooms or zones are unoccupied for a certain length of time with photo sensors[12].

In this dissertation, we tackle the integrated services which produce added value by orchestrating two or more appliances.

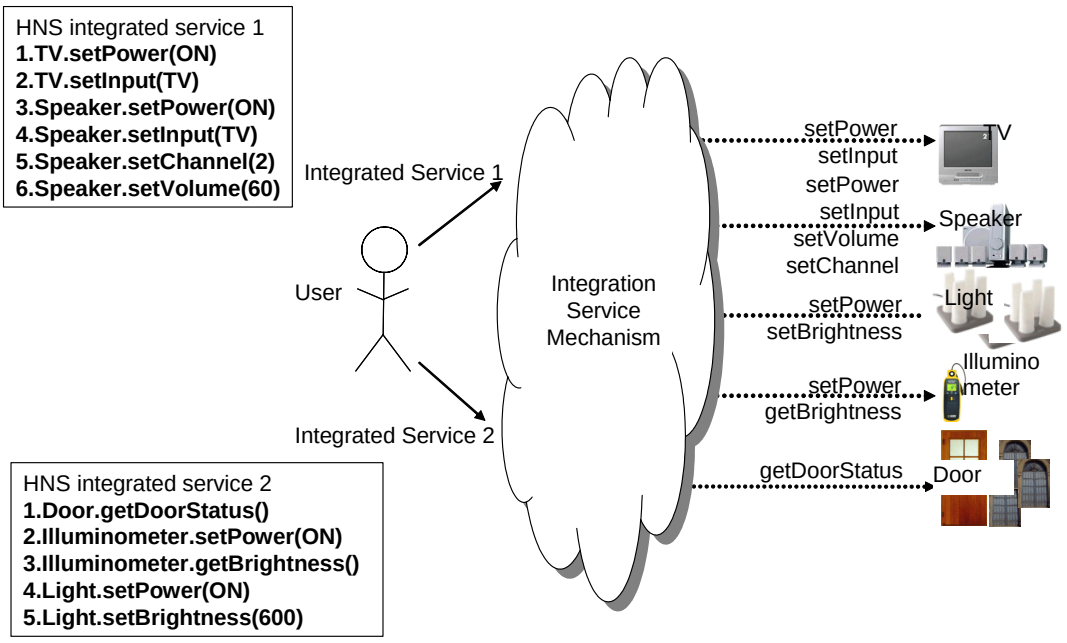


Figure 2.1. Example scenarios of the HNS integrated services

2.2. Practical Scenarios of the HNS Integrated Services

Figure 2.1 shows example scenarios of the HNS integrated services. A TV, a Speaker, a Light, an Illuminometer, and a Door (with a sensor) are used for the appliance integration in this case. The contents of the HNS integrated service scenario surrounded by the square, are as follows.

HNS Integrated Service 1: Auto-TV Service - When the user turns on the TV, the TV is switched on, the speaker is set to 2ch mode, and the volume of the speaker is automatically adjusted for the TV mode.

HNS Integrated Service 2: Coming Home Light Service - When the user comes home and opens the door, the light is turned on, and the brightness is automatically adjusted based on the current intensity provided by the illuminometer.

In this example, the execution which is the sequence of each appliance shows the HNS integrated service. In the case of the integrated service 1, `setPower(ON)` of the TV method, which turns ON the power supply of the TV is executed first. Then, other features of appliances are executed in order.

By using practical scenarios, we develop and evaluate the HNS integrated service in this dissertation.

2.3. Assumptions on Networked Home Electric Appliances

We assume that each networked home electric appliance satisfies the following conditions.

Condition C1: Each appliance has device control interfaces that can be accessed by software (e.g., APIs).

Condition C2: Each appliance has a storage to store application software (server and device control application), a processor to execute the application, and a network interface.

These conditions do not impose unrealistic assumptions. We consider that these features are standard for next-generation home electric appliances. As for Condition C1, standards already exist which prescribe a detailed object template for each category of appliances [11, 12]. Condition C2 is justified by a fact that; the price and size of processors/memories are becoming reasonable enough to embed in home appliances. Indeed, some commercial products already exists which involve Web applications, so that the users can configure and control the product from PCs through a Web interface (e.g.,[40, 48]).

Chapter 3

HNS Integrated Services based on Service Oriented Architecture

3.1. Introduction

To cope with such architecture related problems, this dissertation proposes a new method to implement the integrated services based on the *Service Oriented Architecture (SOA)* [18]. The SOA is a system architecture to integrate autonomous distributed components. The components are loosely coupled with each other by strictly-typed interfaces and standardized communication protocols.

In the proposed method, each appliance is divided into two layers: a *service layer* and a *device layer*. In the service layer, the appliance exports its control interfaces as a set of *services*. If a service is executed, then it sends a control command to the corresponding device with a proprietary protocol. Simultaneously, the service *autonomously* executes (uses) other services exported by other appliances. Thus, the appliances are loosely coupled at the service layer *without* any centralized server. This enables more flexible, robust and load-balanced integrated services.

We first design the HNS-SOA with concrete service scenarios. Based on the design, we propose an implementation framework, and implement a prototype system with *Web services* [10, 51]. We also present a graph-based evaluation method of the integrated services. Using thid evaluation this, we conduct a comparative evaluation of the proposed (SOA) and the existing (SCA) architectures, from the viewpoints of reliability, workload, and coupling.

3.2. Definitions

3.2.1 Service Oriented Architecture and Integrated Services

The service oriented architecture (SOA) [18] is a system architecture to integrate different systems distributed over a network with a standard procedure. Each system exports its own features to the network as a unit of *service* (a set of tasks, which is coarser than an object). The internal logic and implementation of the service are self-contained and encapsulated in the system. The system exposes only interfaces of the service in the form of strictly-typed *exported methods*.

A service user executes the remote exported method and gets the desired results. This remote procedure call is performed by a standardized platform-independent framework. Also, once an exported method is deployed, its interface definition is not allowed to change. Therefore, the change in the internal service logic or service implementation platform do not influence the service user. Thus, a loose coupling between the user and the service is achieved. *Web Services* [10, 51] are widely known as a major SOA framework.

Figure 3.1 shows an example of a SOA. A service user with the client application calls an exported method of Service A. Service A is implemented by tightly-coupled objects, which internally invoke another exported method of Service B. In this exam-

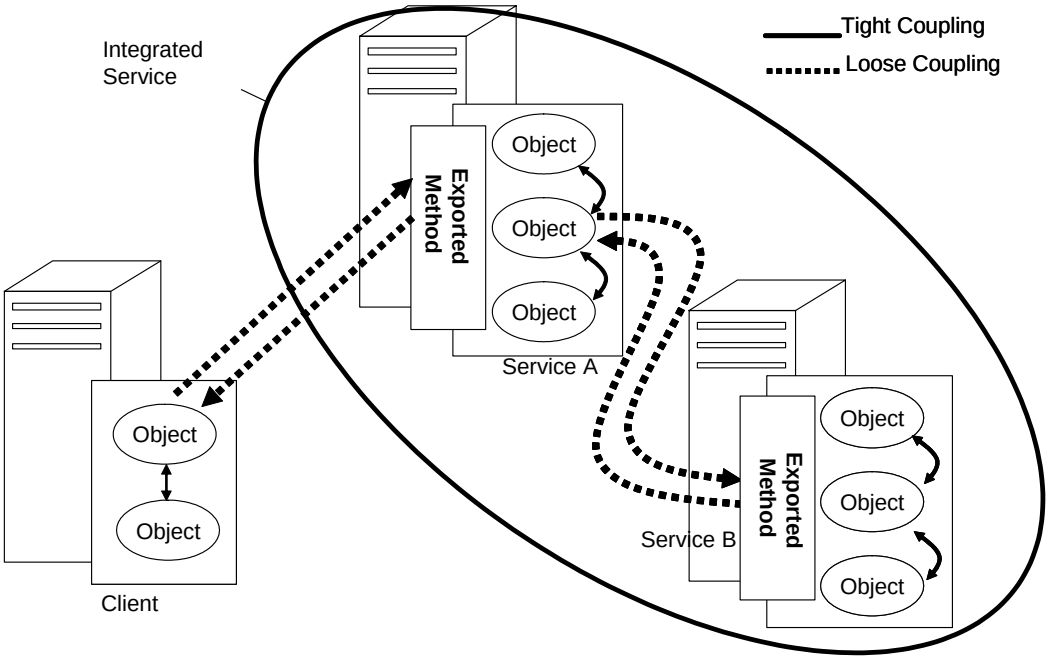


Figure 3.1. Service oriented architecture

ple, the service user uses an integrated service consisting of services A and B, which are depicted by a bold oval.

3.2.2 Scenarios of Integrated Services

For more comprehensive discussion, we introduce example scenarios of the integrated services. In this example, we suppose that a HNS consists of the following nine kinds of appliances (a DVD player, a TV, a speaker, a light, an illuminometer, a door (with a sensor), a telephone, an air-conditioner and a thermometer). We also assume that one appliance exists for each kind, and that the total nine appliances are installed in the

same room.[1]

We prepare the following eight *service scenarios* (denoted by SS_i ($1 \leq i \leq 8$)). These scenarios are determined based on the actual HNS products [19, 42].

SS_1**:** The brightness of the light is automatically adjusted with the illuminometer based on the current intensity of illumination.

SS_2**:** If the user enters a room from the door with a door sensor, the light is turned on.

SS_3**:** When the user turns on the DVD player, the light becomes dark. Then, the TV and the speaker start in the DVD mode.

SS_4**:** When the user watches the TV, the speaker is turned on.

SS_5**:** If the telephone rings while the user is watching the TV, then the volume of the speaker becomes lower.

SS_6**:** The air-conditioning is optimized based on the thermometer.

SS_7**:** If the user enters the room, the air-conditioner starts and adjusts the temperature to a comfortable degree.

SS_8**:** When the user goes out or goes to bed, all the appliances are shut down, and the door is securely locked up.

3.3. Design of Integrated Services

3.3.1 Key Idea

Our key idea is to use SOA to achieve the following issues, which are difficult for the conventional server-centralized HNS.

[1]For multiple appliances of 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.

(A) Standard Communication and Loose Coupling among Appliances: We export features of each appliance as a set of exported methods, which makes the features accessible with the standard protocols in SOA. Thus, the appliances are loosely coupled. This significantly improves the interoperability and extendibility of the HNS. Achieving loose-coupling among components with SOA is not a surprising approach. However, our contribution is to use SOA for the HNS application. For this, we present a concrete appliance structure and implementation framework.

(B) Autonomous Orchestration without a Centralized Server: Our application of SOA enables direct communications among appliances without any special servers. Therefore, the orchestration of the appliances, which has been undertaken by the conventional home server, can be distributed to the appliances. We present a method to implement autonomous collaboration among appliances. Specifically, when an exported method of an appliance is executed, the appliance autonomously determines which exported method should be executed next, and triggers a remote procedure call to the other appliance.

3.3.2 Appliance Structure

To achieve the issues (A) and (B) in Section 3.3.1, we need to implement the following features in each appliances.

(A) Exporting Self-Features: This feature encapsulates proprietary device interfaces, and exports the interfaces to a network using a standardized manner.

(B) Controlling Other Appliances: This feature autonomously invokes interfaces of other appliances, according to a given service scenario.

To implement these features, we divide each appliance into two layers: a *device layer* and a *service layer*, as shown in Figure 3.2.

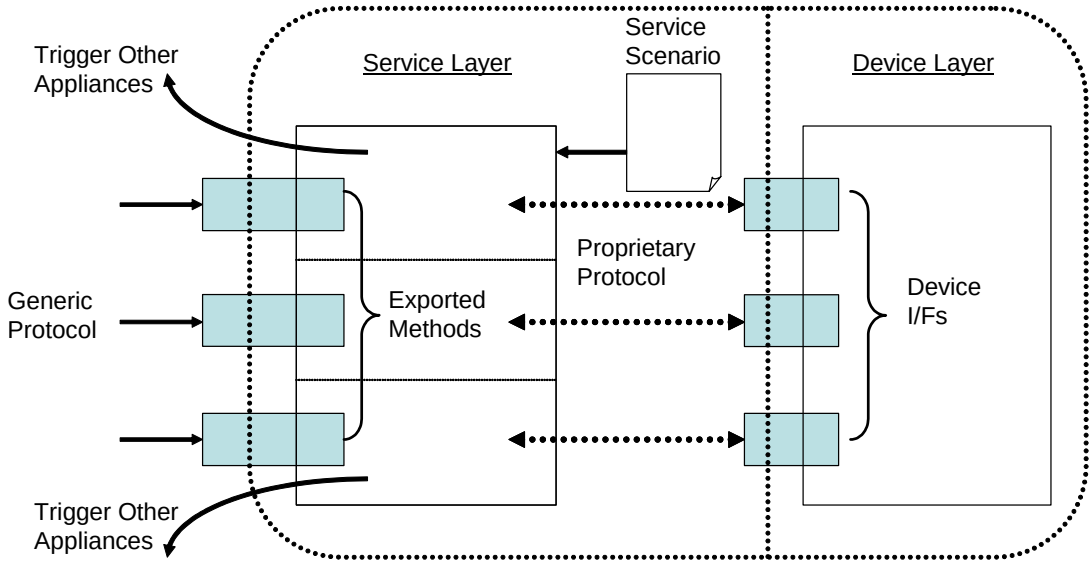


Figure 3.2. Architecture of each home appliance

The device layer refers to the hardware portion (including middle-ware) of the appliance. According to Condition C1, each appliance can be controlled by a software application via a set of device interfaces. Note that the control method is based on a proprietary procedure (or protocol) that the appliance conforms to (e.g., ECHONET for sensors and lights [12], IEEE1394 or UPnP for digital Audio/Visual appliances [50]).

On the other hand, the service layer *wraps* the appliance-specific device interfaces and exports them to the network. The service layer is our original contribution for the HNS-SOA. We implement the layer as a software application on each appliance conforming to Condition C2.

Specifically, we wrap each of the device interfaces (e.g., for a light device, there should be interfaces for ON, OFF and the illumination setting) in a method in the service layer. Then, we export the methods to the network in a generic manner, which

does not depend on appliance-specific procedures or proprietary protocols. For the method of exportation, we use a generic SOA framework such as Web services (with SOAP/XML and WSDL). Thus, all interfaces are opened to a network as a set of exported methods (i.e., a service), which achieve the above (A).

Furthermore, in each exported method, we implement a mechanism by which the method autonomously triggers other exported methods provided by other appliances. Thus, the appliances are orchestrated at the service layer, and the above (B) is realized. The concrete implementation framework of the service layer will be discussed in the next section.

For instance, we take a light and an illuminometer with Conditions C1 and C2, and try to design the SS_1 (in Section 2.3) based on the SOA. Figure 3.3 shows an example of the design. In the figure, an oval represents a service layer of an appliance. A device layer is depicted by an icon. A solid arrow from Service A to Service B with Label L shows Service A invokes (uses) method L exported by B. A dotted arrow represents a control command from a service layer to a device layer. Also, each method is *indexed* by a number which hierarchically specifies its execution order (the notation follows the one in a UML collaboration diagram [13]).

The service scenario starts when the user executes the exported method `Light.ON`. Then, `Light` service invokes other exported methods `ON` provided by `Illuminometer`. Then, each of `Illuminometer` and `Light` services respectively turns on the device.

Next, `Light` service invokes `Illuminometer.getIllumination`, and the `Illuminometer` service internally gets the current degree of illumination from the device. Then, `Illuminometer` sets the obtained illumination to `Light` by the `setIllumination` method. Finally, based on the current degree, `Light` set the optimized illumination to the light device.

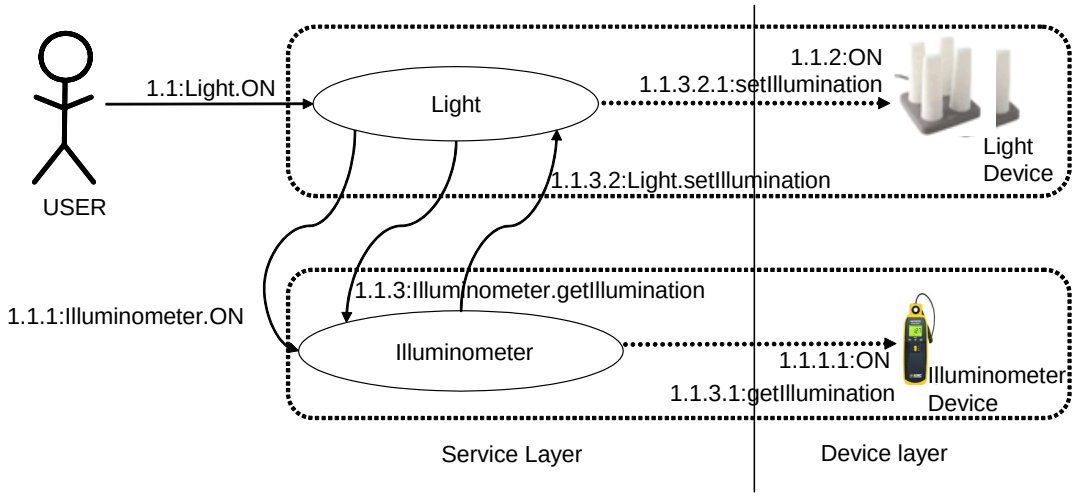


Figure 3.3. Design of service scenario SS_1(Auto Illumination Service)

Thus, we can make appliances autonomously orchestrate at the service layer, which implements the integrated services without the centralized home server.

3.3.3 Service Integration Graph

As shown in Figure 3.3, an integrated service can be characterized by *HNS components* (i.e., the user, services, devices, or a home server) and *use/used relationships* between the components. Hence, we introduce a graph-based notation for the integrated services.

A *labeled directed graph* G is defined by $G = (N, L, E)$, where N is a set of *nodes*, L is a set of *labels*, and $E \subseteq N \times L \times N$ is a set of *labeled directed edges*. For a given integrated service scenario s, a labeled directed graph $G_s = (N, L, E)$ is called a *service integration graph*, denoted by $SIG(s)$, iff G_s satisfies the following conditions:

- N is a set of all HNS components appearing in s,
- L is a set of all methods appearing in s, and
- An edge (p, m, q) exists in E iff p uses method m provided by q.

Next, we extend the service integration graph to the *set* of scenarios. Let $s_1, s_2, ..., s_k$ be a given set of integrated service scenarios. Suppose that for i $(1 \leq i \leq k)$, we have $SIG(s_i) = (N_i, L_i, E_i)$. Then, we define $SIG(\{s_1, s_2, ..., s_k\}) = (\cup_i N_i, \cup_i L_i, \cup_i E_i)$ If $s_1, s_2, ..., s_n$ are all the scenarios in the HNS, then we call SIG $(\{s_1, s_2, ..., s_n\})$ a *full service integration graph*, which is denoted by $FSIG$. Note that any SIG is a subgraph of $FSIG$ by definition.

For example, Figure 3.3 can be regarded as a $SIG(SS_1)$ by mapping each of the ovals and icons to a node, and an arrow to a labeled directed edge.

3.3.4 Designing Integrated Services with Service Oriented Architecture (SOA)

Here we design the eight service scenarios presented in Section 3.2.2 based on SOA. Figure 3.4 depicts an example of a full service integration graph $FSIG$(= $SIG(\{SS_1, SS_2, ..., SS_8\})$) containing the scenarios from SS_1 to SS_8. In the figure, the number appearing with each label corresponds to the actual method described at the left side. Due to limited space, directed edges with the same method are represented by a single arrow. Each label starts with a *scenario number* i of SS_i $(1 \leq i \leq 8)$. The numbers following the scenario number hierarchically specify the *execution order* of the method in SS_i.

Take the scenario SS_4 for instance. In Figure 3.4, we can see a possible design of SS_4 by traversing arrows prefixed by "4.". When the user first turns on the TV (`TV.ON`), the TV service autonomously collaborate with the Speaker service, and sets

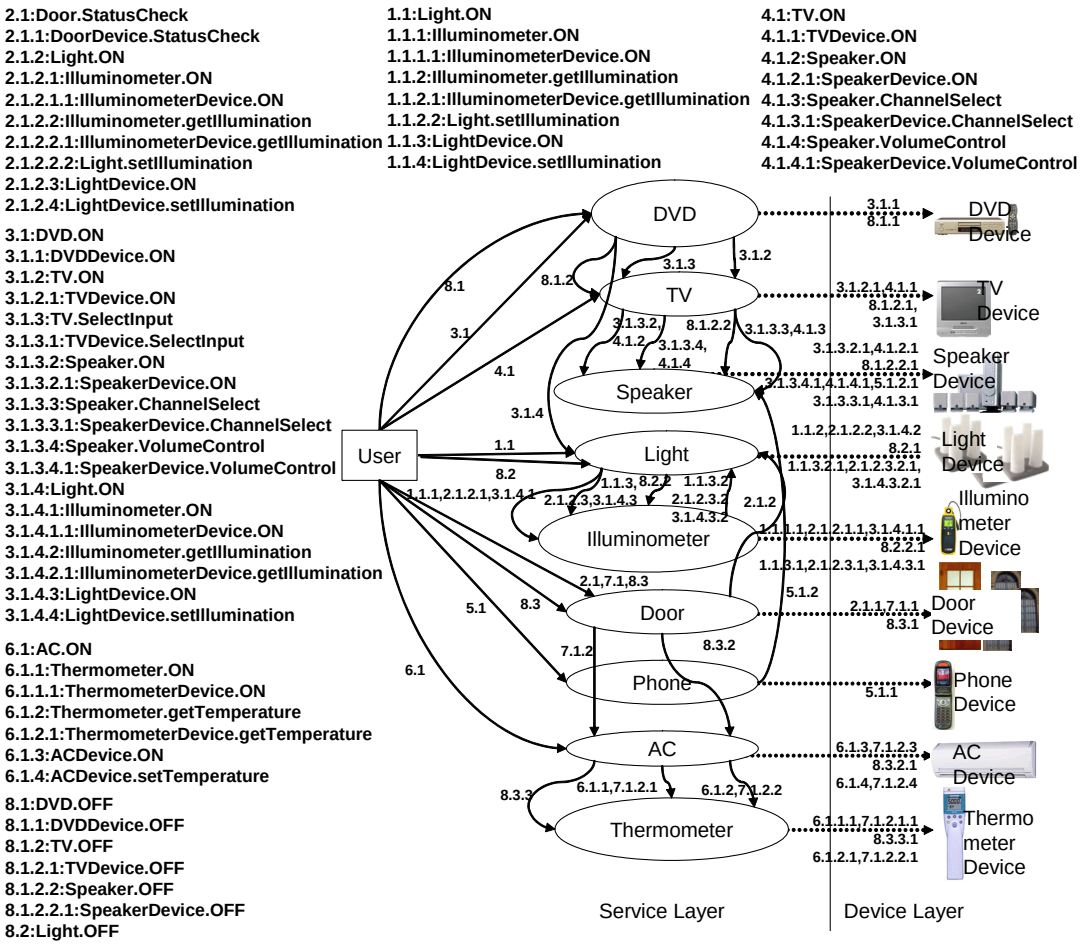


Figure 3.4. Design of integrated services using SOA

the speaker volume and channel. Also, we can see that $FSIG$ in Figure 3.4 contains $SIG(SS_1)$ in Figure 3.3 as a subgraph. SS_3 can be designed by reusing scenarios SS_1 and SS_4. The user first turns on the DVD by `DVD.ON`. Then, the DVD service successively invokes `Light.ON` and `TV.ON`. Next, the Light and TV services respectively execute the same scenarios as SS_1 and SS_4, which completes SS_3.

In the following, we use the HNS-SOA to denote the proposed HNS that extensively exploits SOA to achieve the autonomous and distributed collaboration of appliances.

3.3.5 Designing Integrated Services with Server Centralized Architecture (SCA)

For the purpose of comparison, we try to design the integrated services with SCA. In this approach, a home server sends control commands to the end appliances with the proprietary application and protocol [19, 31]. The home server directly communicates with the communication interface of each appliance. Hence, the service layer is not needed especially for each appliance. Instead, to orchestrate appliances with different network protocols, the home server must implement a *gateway* mechanism for the protocol conversion. Therefore, the implementation of the server tends to be more complicated.

Figure 3.5 shows an example design. In this example, each SS_1 to SS_8 is implemented as an application object which is tightly coupled with the gateway and other objects. Each object sends/receives control commands through the gateway to/from the appliances involved in the scenario. For example, when the user triggers SS_3, the server application executes the object SS_3 to send appropriate control commands to the DVD player, the TV and the speaker.

In the following, we use the HNS-SCA to denote the conventional HNS where the

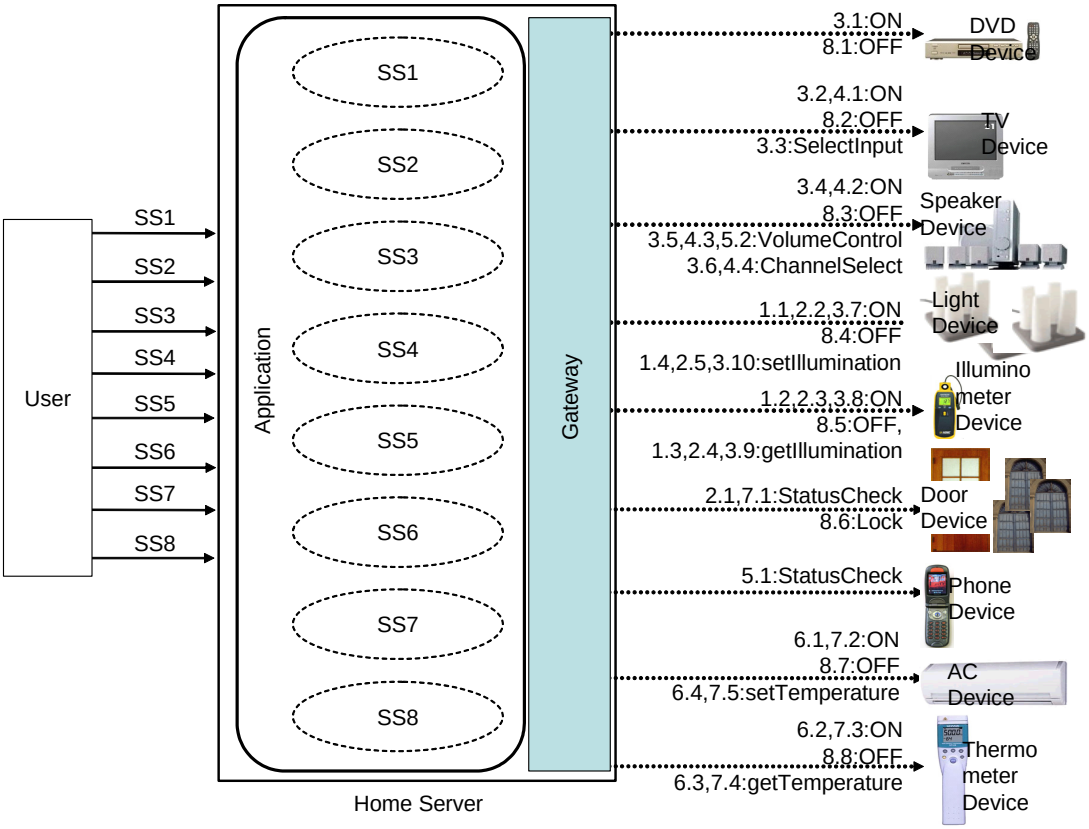


Figure 3.5. Design of integrated services using SCA

centralized home server controls all of the appliances.

3.4. Implementation

3.4.1 Implementation Framework for Service Layer

This subsection presents an implementation framework for the proposed service layer. By definition, a service integration graph $SIG(s)$ for a scenario s is equivalent to the UML collaboration diagram. Therefore, a SIG can be described as a sequence diagram

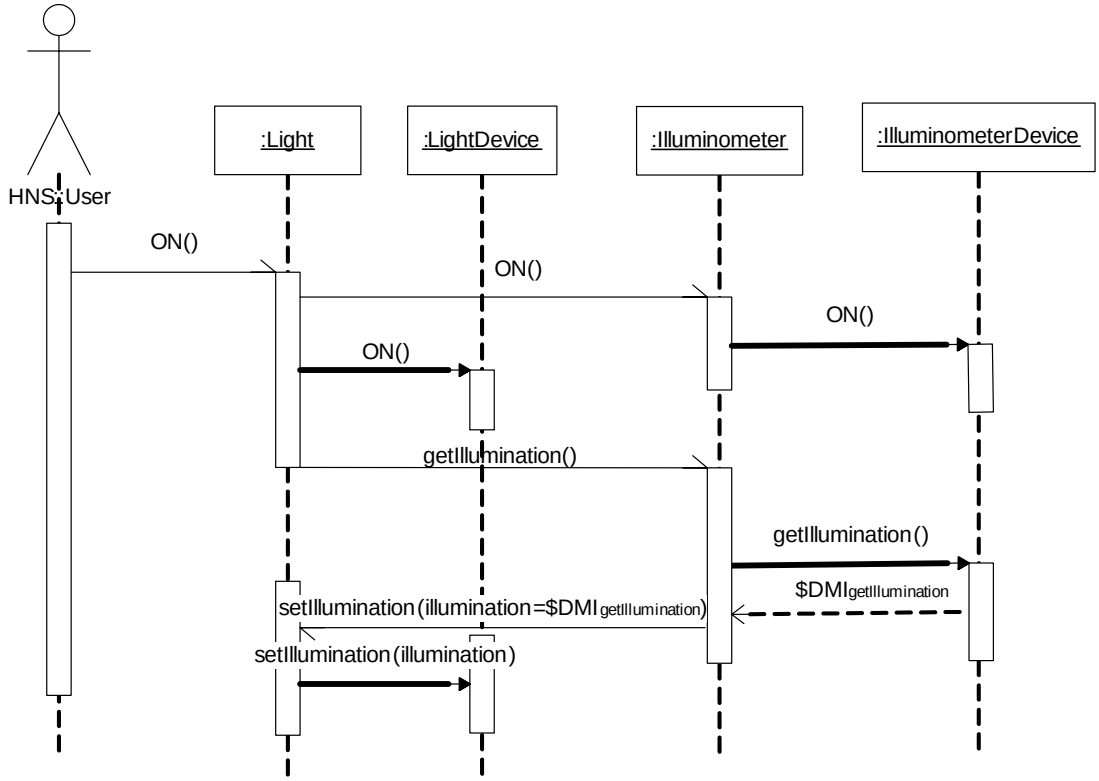


Figure 3.6. Sequence diagram for SS_1

[13]. For instance, $SIG(SS_1)$ in Figure 3.3 can be represented by a sequence diagram as in Figure 3.6. In this figure, we can see that for the autonomous integration of the service layer, the following two types of *method invocations* must be implemented in each exported method.

DMI (Device Method Invocation): DMI refers to the processing of the service layer to send a device control command to the corresponding device layer. According to the proposed appliance structure (see Figure 3.2), there exists a single DMI for every exported method. DMI is constantly executed by the exported method regardless of the service scenario performed.

SMI (Service Method Invocation): SMI refers to processing of the service layer to invoke remote methods exported by other services. Several SMIs are performed before/after the DMI, depending on the current service scenario.

In Figure 3.6, each DMI is represented by a bold arrow, while each SMI appears as a thin arrow. For instance, the `Light.ON` method, which is exported by `Light` service, consists of a single DMI (`LightDevice.ON`), and two SMIs (`Illuminometer.ON`, `Illuminometer.getIllumination`) where the former is executed before the DMI, and the latter is triggered after the DMI.

Our implementation framework of the service layer is as follows. For each control interface d of an appliance, we create an exported method m_d. Then, we code m_d so that m_d invokes d internally, which implements the DMI. If d has a return value, we represent the return value by $\$DMI_d$.

On the other hand, SMI should not be *hard-coded* in m_d, since SMI depends on the service scenarios and can be added or modified by the user later on. Instead, for each appliance we prepare a definition file (called *SMI definition file*) which specifies how the SMIs are invoked in each service scenario. Each appliance looks up the definition file dynamically , and invokes the appropriate SMIs at run time. For the addition or modification of the service scenarios, we just update the SMI definition file without modifying the implementation of the service layer.

Table 3.1(a) and (b) respectively shows SMI definition files for `Light` and `Illuminometer` services as shown in Figure 3.6. Each row corresponds to a single SMI, consisting of the following entries: *context* (a local exported method triggering the SMI), *SSID* (an ID of the service scenario being executed), *service URI* (a URI of the remote service triggered), *pre/post* (before/after the DMI), *methodName* (name of remote exported method), *paramName* (names of parameters of the remote method), *paramType* (types of the parameters), *paramValue* (values of the parameters). When an (local) exported method is triggered in SS_1, each service looks up the table, then dy-

Table 3.1. SMI definition file for SS_1

(a) LightService (http://light.myhome.net/service.jws)

Context	SSID	ServiceURI	Pre/Post	methodName	paramName	paramType	paramValue
ON()	1	http://illuminometer.home.net/service.jws	pre	ON	null	null	null
	1	http://illuminometer.home.net/service.jws	post	getIllumination	null	null	null
setIllumination()	1	null	null	null	null	null	null

(b) IlluminometerService (http://illuminometer.myhome.net/service.jws)

Context	SSID	ServiceURI	Pre/Post	methodName	paramName	paramType	paramValue
ON()	1	http://illuminometer.home.net/service.jws	pre	ON	null	null	null
	1	http://illuminometer.home.net/service.jws	post	getIllumination	null	null	null
setIllumination()	1	null	null	null	null	null	null

namically discovers and performs the appropriate SMI. In this example, the URIs of the two services are assumed to be http://light.myhome.net/service.jws and http://illumino meter.myhome.net/ service.jws, respectively.

Based on the discussion above, the service layer for each appliance can be implemented in accordance with the following *implementation template* (as shown in Figure 3.7):

- The service layer has exported methods so that each method corresponds to a control interface in the device layer.

- Each exported method implements `deviceMethod()` for the DMI. Moreover, `preProcess()` and `postProcess()` are implemented respectively before and after `deviceMethod()`. `preProcess()` and `postProcess()` are routines that dynamically perform SMIs before and after the DMI according to the SMI definition file. These are commonly shared by all the exported methods in the service layer.

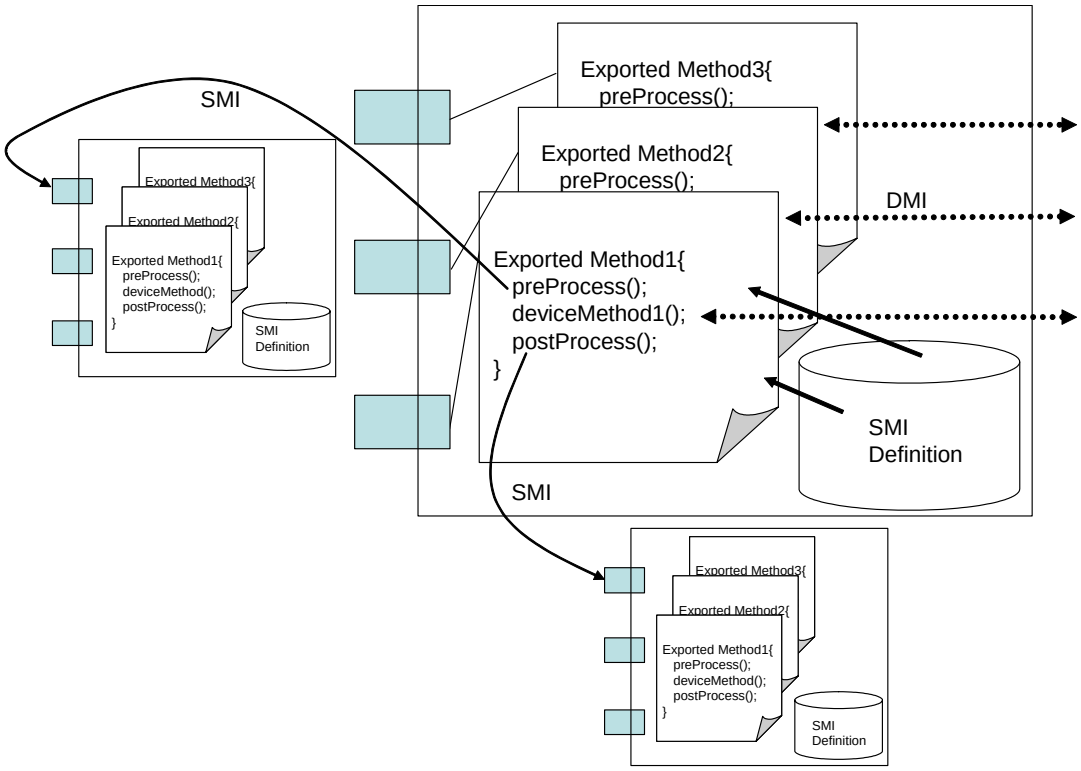


Figure 3.7. Implementation template for the service layer

For instance, consider Figure 3.6 and Table 3.1. When the user invokes the `Light.ON()` method, its method executes `preProcess()` and looks up pre-SMI from Table 3.1(a). As a result, `Illuminometer.ON()` is found and executed. Next, `Light.ON()` sends an `ON` command to `LightDevice` as a DMI. After that, it executes `postProcess()` and looks up post-SMI from Table 3.1(b). As a result, `Illuminometer.getIllumination()` is invoked.

The `illuminometer` service performs SMI with Table 3.1(b). `Illuminometer.ON()` has no SMI processing. `Illuminometer.getIllumination()` executes a post-SMI `Light.setIllumination()`. For this execution, a return value of the DMI ($\$DMI_{getIllumination}$) is passed to a parameter `illumination`. This achieves a

service scenario SS_1, which sets the illumination of light based on the current intensity obtained by the illuminometer.

3.4.2 Prototype System with Web Services

Based on the proposed implementation framework, we have implemented a prototype system. We exploited *Web Services* as a means of service deployment/exportation of the service layer. The prototype was developed under the following environment:

Web server: Jakarta Tomcat 4.1.18
SOAP library: Apache-AXIS 1.1
Language: Java2 SDK SE 1.4.1_02

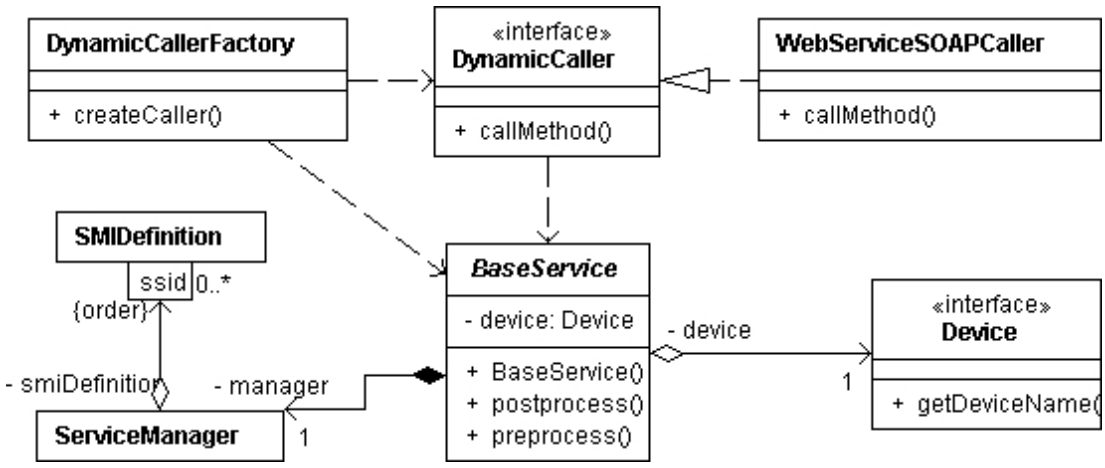
Also, we implemented each device layer as a virtual device. The class diagrams of the prototype system are shown in Figure 3.8. As seen in Figure 3.8(b), every service commonly inherits `a BaseService` class, which

1. Interprets the SMI definition file and dynamically determines SMIs processed in `preProcess()` or `postProcess()`, by using the `ServiceManager` class and

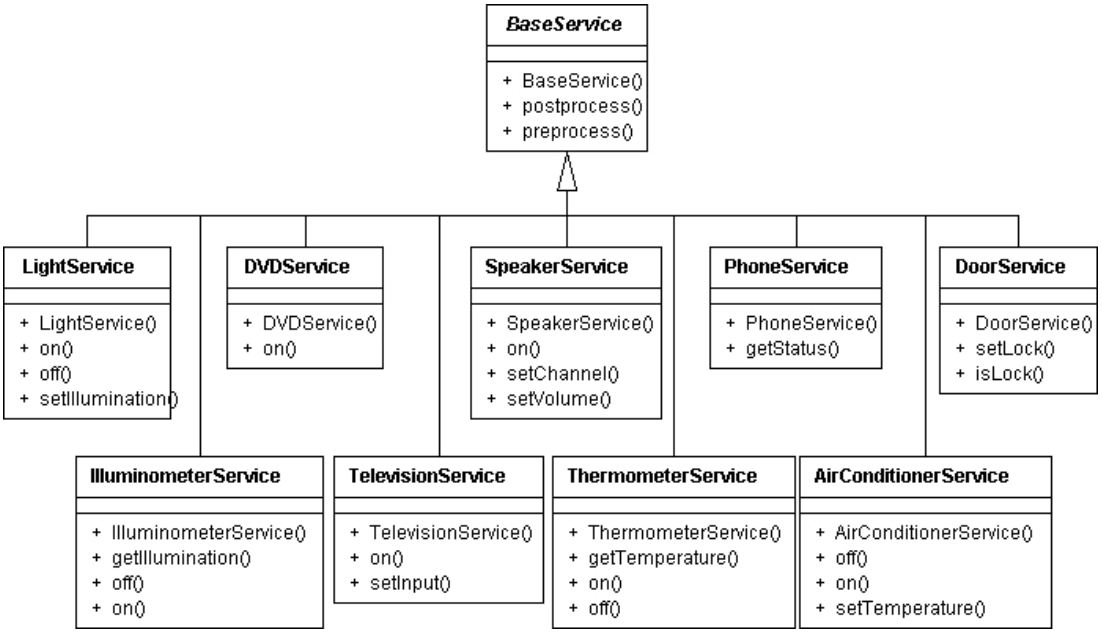
2. Invokes remote WebService for the corresponding SMI, using `a DynamicCallerFactory` class.

Next, based on the service integration graph shown in Figure 3.4, we created an SMI definition file for each service, and uploaded the files to the prototype system. As a result, we confirmed that all the integrated service scenarios SS_1 to SS_8 are executed correctly.

An SMI definition file for a service s can be automatically generated by analyzing the given service integration graph, specifically, examining incident edges of the node



(a)Base classes



(b)Service classes

Figure 3.8. Class diagrams of the prototype system

s and their execution order. If the user wants to add or modify the service scenario, the user just uploads an updated definition file to each appliance. To upload, there is no need to restart the Web server. Also, re-configuring or re-compiling the application itself is not necessary.

3.4.3 Roles in the Proposed Framework

In this section we discuss the roles of the user and of the vendors of appliances in the proposed framework.

First, we assume that the service applications (at the service layer) of each appliance should be developed by the vender, in accordance with the proposed implementation framework. To do this, the vender does not need to be concerned about how the applications are used by other appliances. Instead, the vender has to specify strictly-typed exported methods, and use a generic SOA framework such as Web services for the service deployment/exportation. By using SOA, appliances are loosely coupled, which enables flexible extension and modification of new appliances.

On the other hand, the service scenario development is supposed to be done by the user (Of course, the vender can pre-install the typical default scenarios). By our implementation framework, however, the service scenarios are completely separated from the implementation of the service layer. Hence, the user creates the service integration graph, derives SMI definition files, and then uploads the files to appliances. Thus, the user can easily develop integrated services consisting of any combinations of appliances.

To create the service integration graph, the user needs to know the detailed definitions of exported methods of appliances. This can be supported by *an integrated service creation environment*, which exploits SOA's service discovery techniques, such as WSDL and UDDI of Web services. We are currently developing the tool support for

the creation environment. Note that the addition or modification of the service scenarios are not easy in the conventional HNS-SCA (see Figure 3.5), since the user needs to update the applications of the home server. It is generally difficult for the generic users to develop control applications. The users can only configure the setting of the ready-made applications, which significantly limits the flexibility and extendibility of the service scenarios.

3.5. Evaluation

In this section, we quantitatively evaluate the proposed HNS-SOA from several architectural aspects. Specifically, for the service integration graph presented in Section 3.3.3, we define three kinds of metrics: *reliability*, *workload*, and *coupling*. Then, we conduct a comparative study with the conventional HNS-SCA.

3.5.1 Reliability

Assuming that each HNS component may fail, we evaluate the system-wide reliability of HNS from the viewpoint of the availability of the integrated services. For a given HNS with integrated service scenarios, we define *n-reliability*[17] as the probability that at least n service scenarios are operational in the HNS. The n-reliability varies depending on the HNS architecture as well as on the reliability of each (single) component.

To compute n-reliability, we apply the *Sum of Disjoint Products (SDP) approach*[17, 44, 49] to the service integration graph. The SDP is a method to derive network reliability based on a path-set and cut-set of the graph theory. Intuitively, when a graph G and the reliability of each node (and edge) are given, the SDP method calculates reliability such that at least one of specified set of subgraphs of G is operational, by

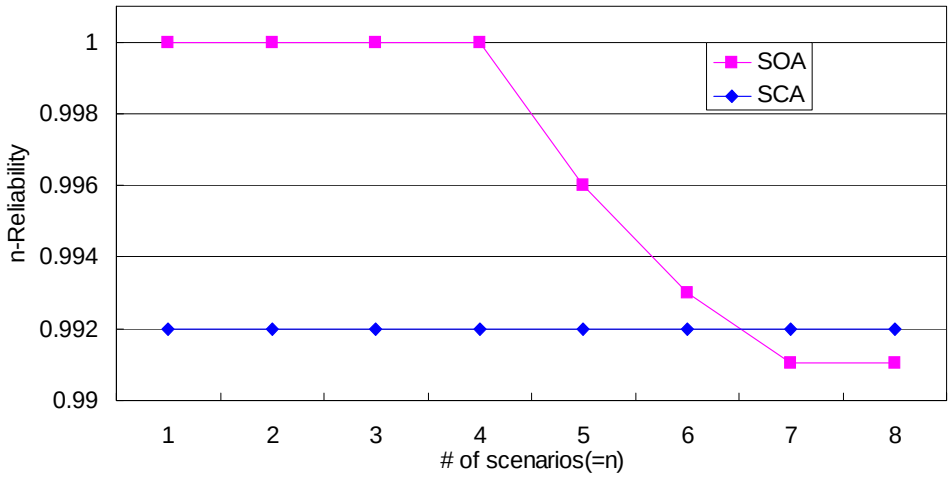
taking the overlaps among the subgraphs into account.

As discussed in Section 3.3.3, each service scenario s can be characterized by $SIG(s)$, and a SIG is a subgraph of $FSIG$. Hence, n-reliability can be computed by SDP in such a way that some n SIGs are operational in $FSIG$. For instance, in Figure 3.4, 1-reliability is calculated by SDP as a probability where at least one of $SIG(SS_1)$, ..., $SIG(SS_8)$ is operational. Similarly, 2-reliability is derived from $SIG(\{SS_1, SS_2\})$, $SIG(\{SS_1, SS_3\})$, ..., $SIG(\{SS_7, SS_8\})$. Thus, taking all combinations from the given set of scenarios, we can compute n-reliability with the SDP method.

To evaluate the reliability purely relevant to the architectural differences (see Figures 3.4 and 3.5), we do not consider any faults in the network or in the devices of each appliance. Hence, we assume that only the services (in HNS-SOA) and the home server (in HNS-SCA) may fail. Although estimating the reliability of each HNS component is generally difficult, we suppose that the reliability of each service in HNS-SOA is uniformly 0.999. As for HNS-SCA, assuming that the home server consists of eight tightly-coupled objects, we set its reliability at $0.992(= 0.999^8)$ (Actually, the reliability is expected to be lower than that, since we should consider the reliability of the gateway, strictly speaking).

We have applied the SDP method to two service integration graphs in Figures 3.4 and 3.5. The result is shown in Figure 3.9. In the figure, the horizontal axis represents the number of service scenarios, while the vertical axis plots n-reliability.

From the result, it can be seen that n-reliability for the HNS-SCA becomes equal to the reliability of the home server because all service scenarios depend on the centralized server. In other words, if the home server fails, all the scenarios become unavailable. On the other hand, in HNS-SOA, the eight scenarios are executed by the distributed services. Hence, even if some services crash, some scenarios can be partially operational. Thus, the HNS-SOA achieves a higher fault tolerance than the HNS-SCA.



	n	1	2	3	4	5	6	7	8
Architecture Type	SOA	0.99999	0.99999	0.99999	0.99998	0.996	0.99302	0.99104	0.99104
	SCA	0.992	0.992	0.992	0.992	0.992	0.992	0.992	0.992

Figure 3.9. n-reliability

For $n = 7, 8$, HNS-SCA becomes slightly more reliable than HNS-SOA. Since HNS-SOA contains more components than HNS-SCA, the probability that *all* the components in HNS-SOA are operational becomes smaller than that of HNS-SCA. Thus, a trade-off relationship between the fault-tolerance and the probability that all systems are green exists.

3.5.2 Workload

We try to estimate the *workload* of each HNS component (i.e., the service layer in HNS-SOA and the home server in HNS-SCA) imposed by the integrated services. The workload varies depending on the appliances involved in the integrated services and

Table 3.2. Workload

Element	*WL*
DVD	10.7
TV	26.1
Speaker	29.8
Light	57.4
Illuminometer	57.4
Door	18.7
Phone	3.7
AC	18.1
Thermometer	18.1
StandardDev	17.925

Element	*WL*
Home Server	86.3
StandardDev	86.3

the *usage frequency* of the scenarios.

Suppose that $FSIG = (N, L, E)$, $SIG(s_i)$ $(1 \leq i \leq n)$ for scenarios s_i's, and usage frequency f_i of scenario s_i are given. Now we define an appearance function $c_i : N \rightarrow \{0, 1\}$ such that for $v \in N$, $c_i(v) = 1$ if v is contained in $SIG(s_i)$, otherwise, $c_i(v) = 0$. The function c_i checks if the node (component) is used in scenario s_i. Then, for each component $v \in N$, the workload of v, denoted by $WL(v)$, is defined as:

$$WL(v) = \sum_{i=1}^{n} f_i \times c_i(v)$$

To obtain the usage frequency, we interviewed 12 users. We asked them the estimated usage frequency of SS_1 to SS_8 per week, and obtained the average number of usage of each scenario. Based on this, we calculated the workload of the service layer of each appliance (in HNS-SOA) and the home server (in HNS-SCA).

Table 3.2 shows the result. The columns show the workload imposed to each service layer (or the home server) and the standard deviation. The workload is concentrated on the home server in HNS-SCA. In HNS-SOA, although the `Light` and `Illuminometer` services suffer from a relatively large workload, these services are still smaller than that of the home server.

Next, we consider the *load-balancing* schemes. In HNS-SCA, the load is con-

centrated on the home server only, and the server itself has to be load-balanced, for instance, by deploying a secondary server. As a result, the load-balancing is so global and inefficient that even unused services are load-balanced. On the other hand, HNS-SOA is more efficient in the sense that only some services with heavy workload need to be taken care of. For instance, in Table 3.2, if the user selects `Light` and `Illuminometer` services for the load-balancing, only these services should be duplicated. Thus, a more flexible and local load-balancing scheme can be applied to HNS-SOA.

3.5.3 Coupling

The coupling is a metric to estimate how strongly a HNS component *relies* on (or is on relied by) the other components. If a component v with extremely high coupling is broken or modified, many other components dependent on v are influenced, which prevents the integrated services from working correctly.

For a given $FSIG = (N, L, E)$, we define a coupling of node $v \in N$ as the number of nodes connected to/from v. More specifically, for $v \in N$, let $use(v) = |\{v' | \exists m; (v, m, v') \in E\}|$ (i.e., # of components that v uses), and let $used(v) = |\{v' | \exists m; (v', m, v) \in E\}|$ ((i.e., # of components that use v)). Then, the coupling of v is defined by $coup(v) = use(v) + used(v)$. For example, let us consider the *TV* service in Figure 3.4. In that case, $use(TV) = |\{Speaker, TVDevice\}| = 2$, and $used(TV) = |\{User, DVD\}| = 2$. Hence, $coup(TV) = 4$.

We compute the coupling for each component in Figures 3.4 and 3.5. Table 3.3 summarizes the result. The coupling values of all services in HNS-SOA are well-balanced. Also, the components in HNS-SCA are heavily concentrated on the home server. This concentration implies that the crash of the server is fatal for all integrated services, as discussed in Section 3.5.1.

Moreover, let us consider the density of each coupling (i.e., each edge in the ser-

Table 3.3. Coupling

	coup	
Element	***use***	***used***
DVD	3	1
TV	2	2
Speaker	1	2
Light	2	3
Illuminometer	1	1
Door	3	1
Phone	2	1
AC	2	2
Thermometer	1	1
HS	9	8

vice integration graph). In HNS-SOA, services are loosely coupled. Hence, even if internal implementation or device control procedures of an appliance change, the other appliances are not affected, as long as the type definition of the exported method does not change.

However, in HNS-SCA, the home server and each appliances are tightly coupled. Therefore, changes in either the server or the appliance cause a significant decline in the interoperability. For example, when adding appliances that conform to a new device protocol to the existing HNS, we have to update the gateway implementation of the home server. This update significantly influences all the existing appliances, which is serious enough to cause a decline the interoperability among the new and existing appliances. Taking these into account, it is considered that the coupling is much stronger than the values in Table 3.3.

3.6. Discussion

3.6.1 Advantage and Limitation

The advantages of the proposed framework are summarized as follows:

(1) Since the appliances are loosely coupled at the service layer with the SOA, the interoperability between appliances is improved. This improvement facilitates addition and modification of the appliances.

(2) Since the integrated services are realized by autonomous collaboration among the appliances, the proposed framework does not need the centralized server. As a result, the integrated services become more fault-tolerant and load-balanced.

(3) Due to the proposed implementation template, the implementation of the service layer and the service scenarios are well separated. Therefore, the user can easily add or modify the integrated service scenarios.

We examined the relationship between SOA and each of these advantages. First, the above (1) is achieved by the nature of SOA, which is not limited to the HNS domain. The above (2) is due to the proposed service layer which extensively uses the merit of the loose coupling. We have implemented mechanisms of the exportation of the self-features as well as implementing the control of other appliances (see Section 3.3.2) in the service layer. These feature allow us to decentralize the service orchestration task among the appliances themselves. The implementation template in (3) makes full use of the characteristics of the SOA so that we can use any feature of the appliance uniformly as an invocation of the exported method. With this advantage, the service layer for every appliance has a common structure, where the control of other appliances and the exportation of the self-features are achieved as SMI and DMI, respectively. Also, the proposed implementation template looks up the content of SMI from the

external definition file. This allows the separation of the service scenarios from the implementation of the service layer.

Of course, the proposed framework is not perfectly superior to the conventional one. As a drawback to the fully-distributed control of the appliances with the SOA, the following issues are currently anticipated:

Cost of Appliances: Each appliance must be intelligent enough to satisfy Condition C2 (see Section 2.3), in order to realize the service layer. This requirement makes the cost of appliances more expensive than in the conventional method.

Communication Overhead: It is expected that the communication overhead required for the service orchestration cannot be ignored. Hence, when applying to the integrated services that require a *hard-realtime* response, we need careful consideration of implementation.

Global Management: Since the service control is fully distributed, managing all the appliance at once is difficult. Hence, more sophisticated mechanisms are required for detection of faulty appliances and the application of a global security policy, etc [47].

3.6.2 Related Work

BPEL4WS [7, 53] is known as a standard service orchestration framework. BPEL4WS is an XML-based language describing new services that integrate the existing distributed service components. Using BPEL4WS as an alternative of the service integration graph in the proposed framework might be possible. Indeed, in [29], a service oriented method to orchestrate intelligent appliances using BPEL4WS is presented. However, the existing BPEL4WS platform needs the orchestration server (typically

called BPEL engines), which follows a server centralized architecture. Therefore, the existing framework cannot be directly used for implementing the proposed HNS-SOA.

In [45],distributed networked appliance architecture “AMIDEN” is proposed. By unitizing each function embedded in a networked appliance, an autonomous setup to create a multi-functional service is realized. However,since creating scenarios by home users is not assumed, flexible creation of the integrated service scenarios with original appliance functions cannot be performed.

A new language called WS-CDL (Web Services Choreography Description Language) [52] is currently being specified by W3C. The WS-CDL aims to strictly define observable interactions between services from a global point of view. It adopts π-calculus [33] as a mathematical foundation to deal with complex ordering and relationships among services. The application is focused mainly on the on-line transactions among enterprises. The WS-CDL might be used for modeling the integrated services in our HNS-SOA. However in practice, it is difficult for end users to define rigorous relationships among appliances. Also, the current HNS services do not require much complex collaboration. Considering the convenience of users and the complexity of the current integrated services, we do not envision many HNS applications making full use of π-calculus and WS-CDL. Investigation of more sophisticated HNS services and application of WS-CDL is left for our future research.

3.7. Summary

In this Chapter, we have proposed a framework using SOA to design and implement integrated services for home network appliances. We have also conducted a comparative evaluation of the proposed HNS-SOA with the conventional HNS-SCA.

In our future research, we will evaluate the limitation of the proposed HNS-SOA from a more practical viewpoint. For this, we are currently extending our prototype

systems to more appliances and more service scenarios. Based on the evaluation, we plan to investigate the management framework and security schemes which will suit the SOA well.

Chapter 4

Feature Interactions in HNS Integrated Services

4.1. 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 [11][12][37]. 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, for example, group control of appliances [19], health monitoring [46], and home security [31]. 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. HNS is considered count as 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 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 the brightness of the lights, the Coming Home Light service sets a 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. [25]. 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 4.7.7 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 chapter 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 in-*

teraction. 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.

4.2. Preliminaries

4.2.1 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 chapter, 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., [11][12]). Also, the price and size of processors/memories are becoming reasonable enough to embed in home appliances. Some recent products (e.g., [48]) 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 [55], HAVi [20], Jini [23] and UPnP [50]. In this chapter, 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.

4.2.2 HNS Integrated Services

Controlling only a single networked appliance does not offer much added value compared to traditional appliances [25]. 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 the total ten appliances are installed in the same room.[1] We prepare

[1]For multiple appliances in the same kind, we regard them as independent appliances. For example,

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 [19][31].

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 registers 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.

---

if there are four lights in the room, we consider four instances; Light1, Light2, Light3 and Light4.

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. */
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 4.1 shows an example of API sequences for all service scenarios SS_1 to SS_7.

4.2.3 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.

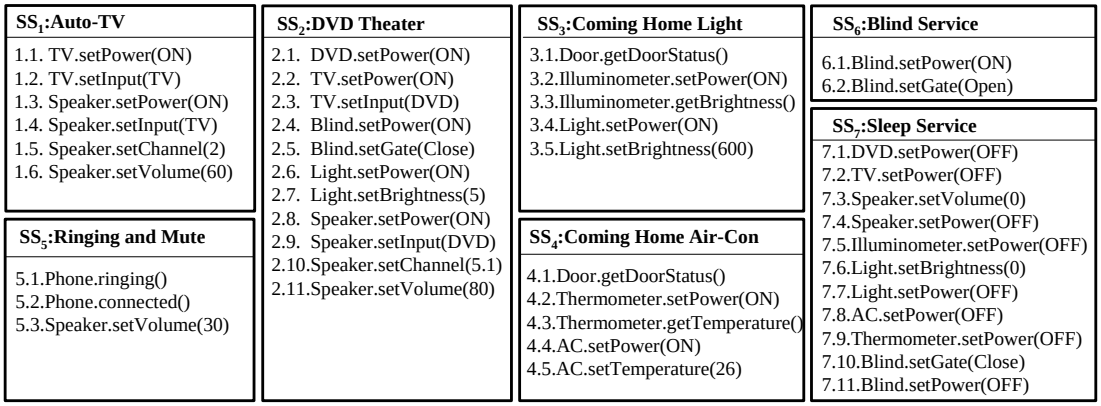
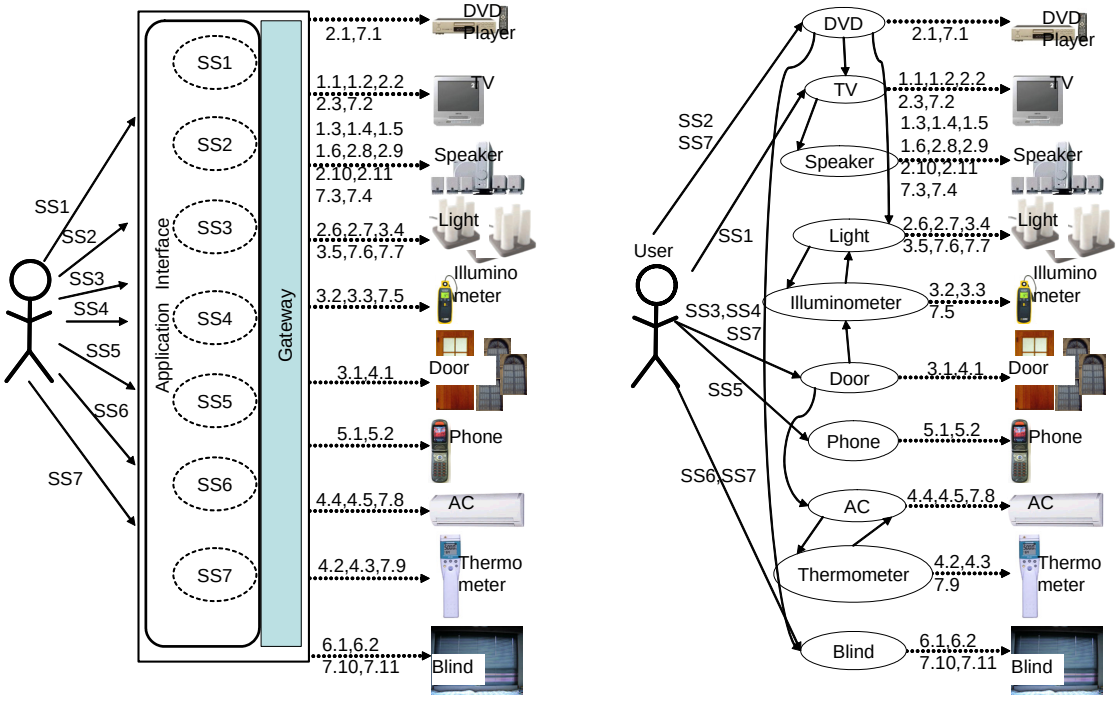


Figure 4.1. API Sequences for SS_1 to SS_7

A straightforward way to orchestrate the appliances is to deploy a powerful *home server* in the HNS [19][29][31]. The home server takes centralized control of all appliances in the HNS, which we call *Server Centralized Architecture (SCA)*. Figure 4.2(a) depicts the HNS-SCA, 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 4.1. 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 [12] for lights, blinds and sensors, and a UPnP for Audio/Visual appliances [50]). Therefore, the home server requires a sophisticated gateway [37] to achieve the interoperability among the appliances.

Delegating the appliance control to the appliances themselves is another way to have appliance orchestration. In Section 3, we proposed an autonomous-decentralized architecture based on the *Service Oriented Architecture* (HNS-SOA) [29], which is shown in Figure 4.2(b). Attaching an application adaptor (called a *service layer*, depicted by an oval) to each appliance, an appliance can autonomously trigger other ap-



(a) HNS-SCA (b) HNS-SOA

Figure 4.2. HNS Architectures for Appliance Orchestration

pliances, 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 integrated 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.

4.3. Formal Definition of HNS

4.3.1 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 [12] [2]. In this chapter, the methods correspond to the APIs discussed in Section 4.2.1. 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 4.1 summarizes an example of properties and types for the ten appliances introduced in Section 4.2.2. 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 `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)`.

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

Table 4.1. Appliance Properties

Appliance	Property	PropertyType
AirConditioner	Power	{ON,OFF}
	TemperatureSetting	unsigned int (℃)
Thermometer	Power	{ON,OFF}
	CurrentTemperature	unsigned int (℃)
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}

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 chapter, 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 (tTemperature Setting 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 4.3.1 (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 $c = [\texttt{Power=='ON'} \wedge \texttt{TemperatureSetting} > 20]$ 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, $\prod_{Power}(c) = [\ \texttt{Power=='ON'}\ ]$, which is a projection of c onto `Power`. Next, we define each networked appliance as follows.

Definition 4.3.2 (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 4.2. 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' Ù 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' Ù CurrentBrightness=*	
Door	getDoorStatus()	Power='ON' Ù 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

Table 4.2 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.

4.3.2 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 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. [25][32]). In this chapter, 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 modeling is because the impact of a method to the environment properties 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 4.3.3 (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 home.

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

Table 4.3. 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

`Brightness:` the current intensity of brightness in the home.

`Volume:` the current sound volume in the home.

Table 4.3 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.setTemperature(...)`). This property implies that setting the temperature of the air-conditioner can write (update) the current temperature degree of the home.

4.3.3 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 4.3.4 (Home Network System) A home network system is defined as $HNS = (D, e)$, where

- $D = \{d_1, d_2, ..., d_n\}$ 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 4.1 and 4.2 and an environment defined in Table 4.3. Next, as mentioned in Section 4.2.2, 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 4.3.5 (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 a *HNS integrated service scenario*.

Thus, the API sequences shown in Figure 4.1 are finally formalized as the HNS integrated service scenarios.

4.4. Feature Interactions in the HNS Integrated Services

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

4.4.1 Appliance Interactions

When multiple service scenarios simultaneously invoke incompatible methods of a common appliance, one method conflicts with another, which results in a feature in-

teraction 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 4.1 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 4.2, 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`] $= \perp$ (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 4.4.1 (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).

4.4.2 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 4.4.2 (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 4.1 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 4.2, `Light.setBrightness()` of SS_3 and `Blind.setGate()` of SS_6 try to write the common environment property `Brightness`. Moreover, `Illuminometer.getBrightness()` of SS_3 reads `Brightness` as well. That is, W_e(`Light.setBrightness()`) $\cap$ W_e(`Blind.set`

`Gate()`) $\cap$ R_e(`Illuminometer.getBrightness()`) = {`Brightness`} $\neq \phi$. Thus, these three methods cause an environment interaction on brightness in the home.

4.5. 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 4.1, 4.2 and 4.3. A set of HNS integrated service scenarios $SS_1, SS_2, ..., SS_7$ shown in Figure 4.1.

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 4.4(a) shows a total of 43 appliance interactions, whereas Table 4.4(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 4.3 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

Table 4.4. 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							

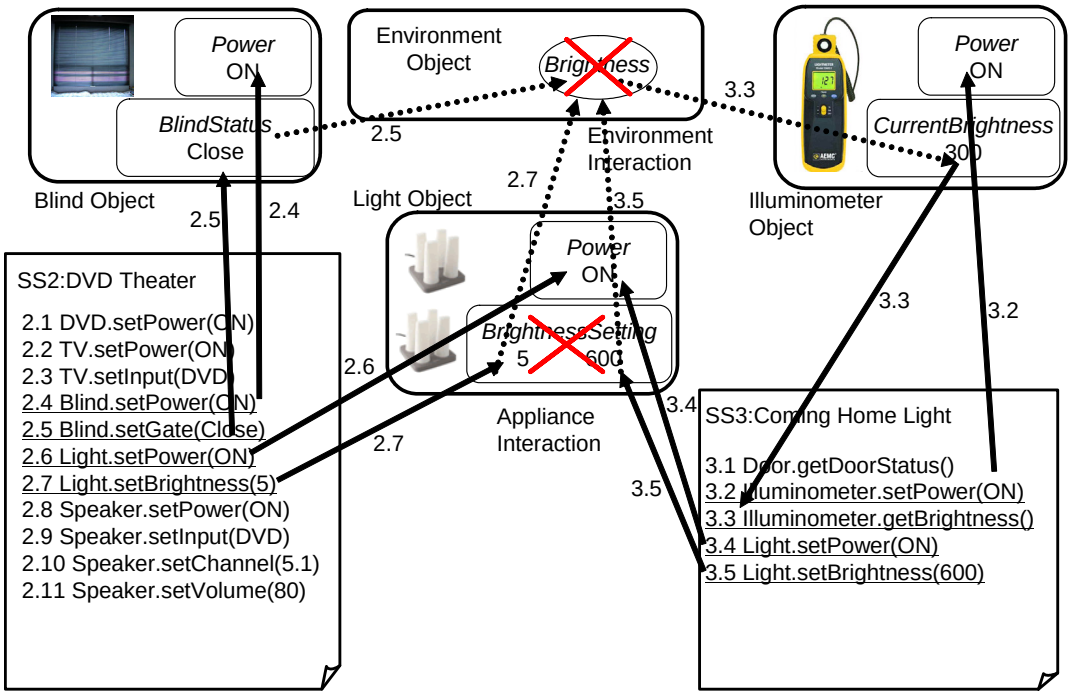


Figure 4.3. Interactions between SS_2 and SS_3

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 4.4.1), which is exactly the same way that FI-(a) was introduced in Section 4.1. 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 4.4.1), by which we can reasonably explain FI-(b) as explained in Section 4.1. Two

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

other environment interactions occur as well between 2.5 and 3.5, and between 2.7 and 3.3.

4.6. Online Interaction Detection

As seen in Section 4.5, 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.

Therefore, we propose an online interaction detection method which is performed during runtime of service scenarios.

4.6.1 Key Idea

In the HNS integrated service, when one of the following conditions are satisfied, feature interactions occur.

(a) In HNS, suppose that for $d.m_i$ contained by ss_i and $d.m_j$ contained by ss_j. Both $d.m_i$ and $d.m_j$ are executed, and one of the conditions D1, D2 are satisfied.

(b) In HNS, suppose that for $d.m_i$ contained by ss_i and $d'.m_j$ contained by ss_j. Both $d.m_i$ and $d'.m_j$ are executed, and one of the conditions E1, E2 are satisfied.

Based on these conditions, an online feature interaction detection is shown, as follows.

STEP 1: The name of the runtime appliance method and service scenario containing its method are acquired.

STEP 2: As condition evaluation and feature interaction detection are performed for a set of appliance methods in a newly executed scenario and in the information of STEP 1.

We prepare an execFlg(execution flag) and a history information per every appliance method for STEP 1. The value of the flag '1' shows under operation of the method and '0' shows that the method is not operating. The history information has detailed information of the executed appliance method (arguments of the method, service scenario ID etc.), and this information is related to the execFlg.

For STEP 2, an application module evaluating conditions of D1, D2, E1 and E2 is implemented. If this module detects interactions, then the feature interaction detection process moves to the process of replying its results or replying with an interaction resolution. If no interaction is detected, the service scenario is executed normally.

We describe concrete designs of online interaction detection for multiple HNS integrated service architectures.

4.6.2 Online Feature Interaction Detection in HNS-SCA

In the HNS-SCA, detecting both appliance and environment interactions can be undertaken by the home server. The home server takes global control of all appliances. Therefore, the home server 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.

Figure 4.4 shows the example of online interaction detection. In this case, SS_2 is under execution, and SS_6 is going to be performed. First, the home server receives the requirements of SS_6 and checks the name of the runtime appliance method in

(1). Next, the information(arguments and service scenario ID, etc) about the runtime appliance method is acquired in (2). Furthermore, condition evaluation and interaction detection are performed for the information on the runtime appliance method and SS_6 in (3). Finally, the appliance interaction ($SS_2 : Blind.setGate(Close)$, $SS_6 : Blind.setGate(Open)$) and the environment interaction ($SS_2 : Light.setBrightness(5)$, $SS_6 : Blind.setGate(Open)$) are replyed to a user as the detection result in (4).

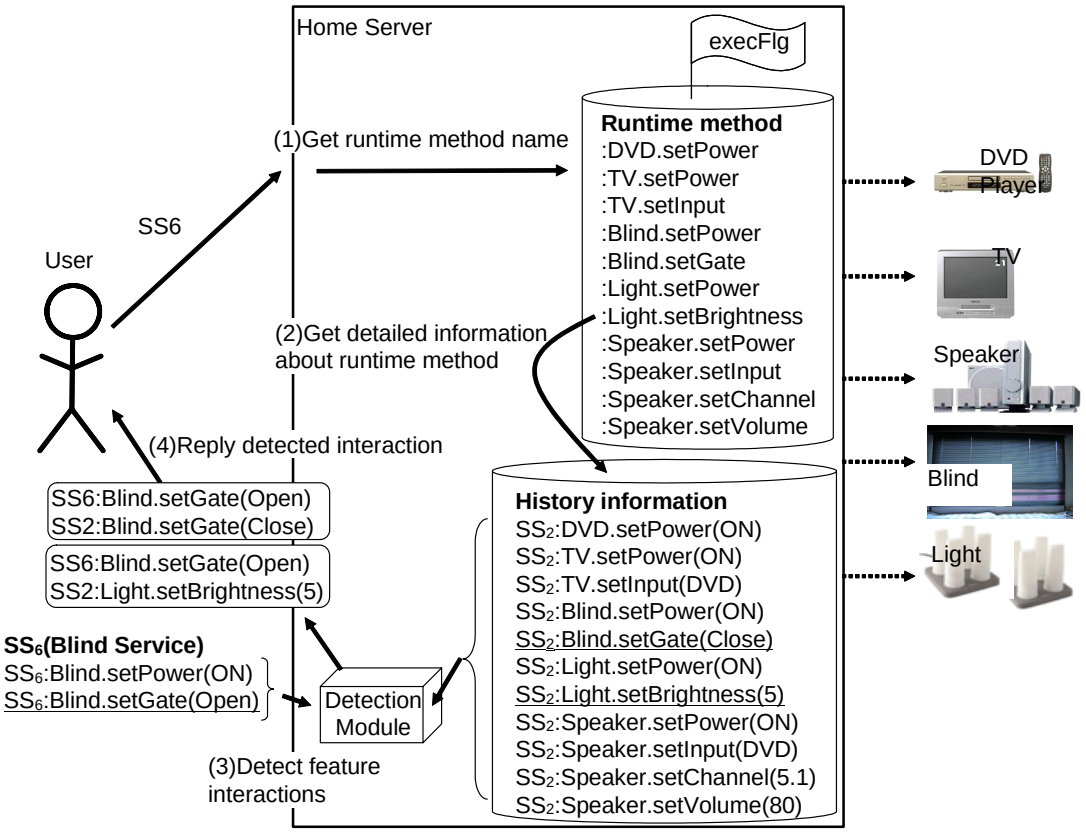


Figure 4.4. Online Feature Interaction Detection in HNS-SCA

4.6.3 Online Feature Interaction Detection in HNS-SOA

HNS-SOA requires a sophisticated mechanism for 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 module for each appliance. Its interaction detection module 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.

In the HNS-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.

We present the detecting process of environment interactions in HNS-SOA.

(Prior setup) For all appliances, offline environment interactions are detected. Every appliance gets to know the combination of appliance-method occurring environment interactions.

(Runtime1:) Whenever a method is executed newly, the running states of all the appliance methods that can conflict with each other are checked.

(Runtime2:) Based on the execFlg and history information, environment interactions are detected by comparing a new invoked method and the runtime method.

Figure 4.5 shows an example of online environment interaction detection. This figure indicates the case where SS_6 is newly performed to SS_2 under execution like Section 4.6.2. In HNS-SOA, the service layer of the Blind receives a requirement of SS_6($SS_6 : setPower(ON)$, $SS_6 : setGate(Open)$) execution directly.

First, the runtime method of Blind is checked in (1). Next, Blind service acquires a history of the runtime method in (2). Then appliance interactions are detected between SS_2 and SS_6 in (3). In this example, when $SS_6 : setPower(ON)$ is executed, no interaction is detected. And in the case of execution $SS_6 : setGate(Open)$, the appliance interaction($SS_6 : setGate(Open)$ and $SS_2 : Blind.setGate(Close)$) are detected and replied to in (4).

The Online environment interaction detection process proceeds after (2). The running states of the appliance methods are checked based on the combination of the appliance method, which can cause environment interactions in (3'). In this case, $SS_6 : Blind.setGate$ and $SS_2 : Light.setBrightness$ are a pair of potential environment interaction. So, the runtime method information of Light Service is required in (4'), and environment interactions ($SS_2 : setBrightness(5)$, and $SS_6 : setGate(Open)$) are detected in (5')(6').

Thus, in HNS-SOA, implementation which is different in each environment interaction and appliance interaction is needed.

4.7. 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.

4.7.1 Rebuild Scenario

Based on the result of the offline detection, rebuild the service scenarios(update , delete etc.) so that any interaction is avoided. This approach is available only in the environment where the services can be flexibly rebuilt. However, the number of interaction also increases with diversification of appliances of service scenarios. Resolving all

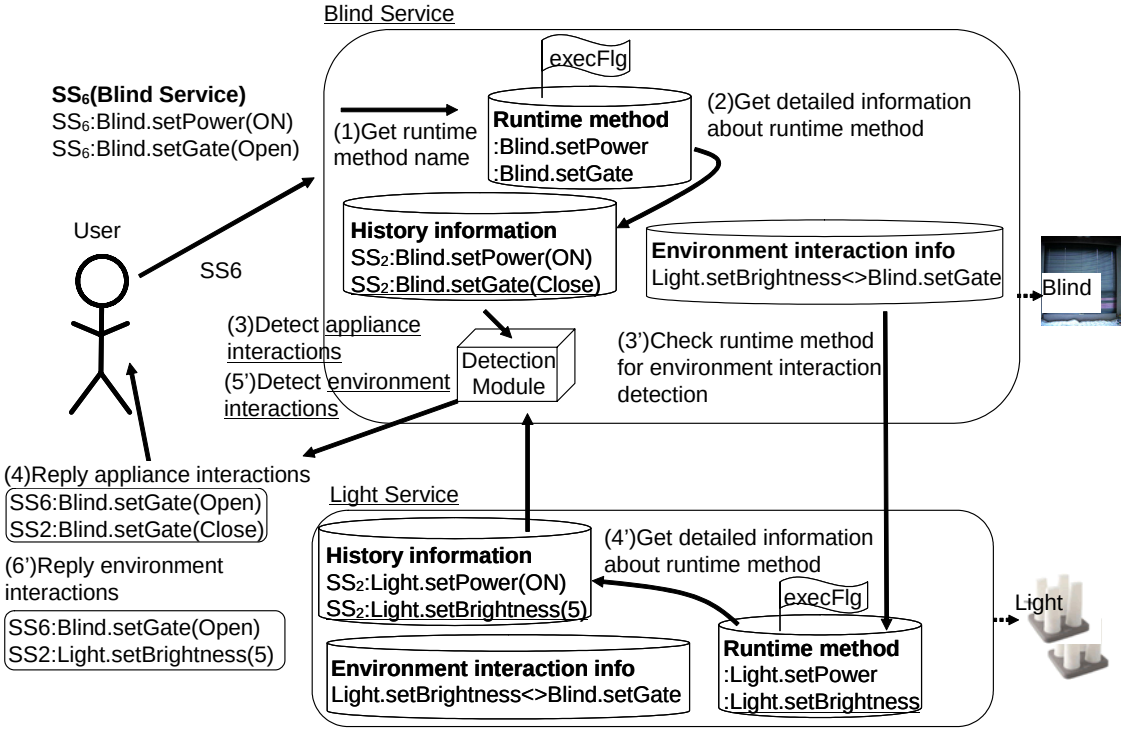


Figure 4.5. Online Feature Interaction Detection in HNS-SOA

potential interactions is difficult for users.

4.7.2 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.

In this approach, we need the following 2 new functions for our system.

(a) Display the results of interactions detection

(b) Dynamic updating and execution of the service scenarios(as we described in chapter 3)

Resolution of interaction is possible only for the integrated services which a user(s) execute. Resolving those interactions of the integrated services which a sensor device(a thermometer and a illuminometer etc.) triggers is difficult for users to do.

4.7.3 Prioritize Services

Assign static priorities to services [25]. If a pair of service scenarios cause an interaction, then all conflicting methods in the service with lower priority are aborted. As prioritizing methods, partial order(between a potential interaction scenario pair) and total order(among all service scenarios) are considered. The partial order needs a large quantity of prioritizing. The total order needs sensitive prioritizing to resolve all interactions. In this approach, the feature interactions which a user cannot control are also solvable(unlike Section 4.7.2). Note in this approach that the execution of the service

with a higher priority is always guaranteed. As we call, the resolution method of a service unit is influenced by interaction resolution sequence.

For example, suppose that 3 service scenarios: SS_a(priority high),SS_b(priority middle), and SS_c(priority low) are given. And SS_b conflicts with SS_a and SS_b. SS_a and SS_c are under execution, and SS_b is newly performed. In this case, the resolution sequence affects the final state of the HNS integrated services.

1. If SS_a and SS_b are evaluated with those interaction conditions first, SS_b cannot be executed. Therefore, the HNS runtime scenarios are SS_a and SS_b.

2. If SS_b and SS_c are evaluated first, SS_b can be executed and SS_c is stopped. Next, SS_a and SS_b are evaluated, and SS_b is stopped. Thus, the HNS runtime scenario includes only SS_a.

4.7.4 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.

Compared with Section 4.7.3, this approach needs more workload for prioritizing. It is remarkably harder for users to give the total order of all appliance methods without using offline detection results. A possibility that the service scenarios will change to what a user does not intend exists.

4.7.5 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.

In this approach, our system needs the following mechanism.

Table 4.5. Comparison of resolution approach

	Resolution Approach	Required Function	Employment Cost	Merit	Demerit
Rebuild Scenario	Offline	Nothing	Scenario rebuilding	No required function	To resolve all interactions is harder
Prompt User	Online	Interaction result display,Runtime scenario updating	Runtime scenario updating in case of interactions occur	Most flexible interaction resolution	Complex required function is needed.Service without user control isn't resolved
Prioritize Services	Online	Interface to prioritize,Runtime automated resolution	Prioritizing services	Automated resolution	It is difficult for users to prioritize,resolution sequence affects final state of HNS
Prioritize Methods	Online	Interface to prioritize,Runtime automated resolution	Prioritizing methods	Automated and flexible resolution per method	It is difficult for users to prioritize. A possibility which the service scenarios change to what a user does not intent exists.
Prioritize Users	Online	Interace to prioritize,Runtime automated resolution	Prioritizing users	Automated resolution Easy prioritizing	A user with lower priority cannot always execute services. Service without user control isn't resolved

(a) An interface which can acquire the user information executing the service scenario.

(b) An interface for prioritizing users and a databse to store them.

(c) An application module to evaluate executing service scenarios with users and resolve their interactions.

The workload of the pre-configuration process(prioritizing users) is low. This approach is also possible only for the services executing by a user.

4.7.6 Hybrid Resolution Method

As shown above, much granularity can be considered for interaction resolution. Each resolution approach has its merits and demerits(Table 4.5). Using multiple resolution schemes together achieves fine interaction management for each of system, service, method and user levels.

In user level resolution, priority settings for parent and child are usable. Services such as a security management which have the highest-priority, need service level pri-

ority management. Method level resolution contributes flexible customizing scenarios on other level resolution. For example, in service level resolution, lower priority scenario can be executed flexibly with method level scenario updating.

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.

4.7.7 Related Work

Compared to the existing method [25], 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 [25], 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 [32]. Within the domain of embedded control systems, their method 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., [15]) and/or structural analysis of rule-based methods (e.g., [56]) would enable efficient pre/post-conditions checking of the appliance methods. A negotiating agent approach [16] would also help to implement an automatic interaction resolution for the scheme (h) in Section 4.7.

4.8. Summary

In this chapter, 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 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, user-view, and so forth. Adaptation of the conventional techniques in telephony to the HNS integrated services is also interesting topic for further study.

Chapter 5

Conclusion

5.1. Achievements

In this dissertation, we addressed two issues in developing the HNS integrated services.

Our first achievement is the HNS-SOA framework. In this research, we have prepared service specifications for practical services. With these scenarios, we designed and implemented the HNS-SOA.

In the proposed HNS-SOA, all appliances have a service layer which enables service integration without a centralized server. In addition, the implementation template for separating the integrated service scenario from implementation of the service layer realized a dynamic change of scenarios, and easy implementing.

Moreover, we evaluated the proposed framework by the graph-based method. By this evaluation, the HNS-SOA has a several advantages compared with the conventional HNS-SCA. In our proposed HNS application framework, features of SOA, such as loose-coupling and autonomous orchestration without a centralized server, improve the HNS integrated services at points of the reliability, interoperability, and load-balancing. Extendibility is also improved by the proposed implementation template

since the implementation of the service layer and the service scenarios are well separated.

A second contribution is the feature interaction detection. We modeled the HNS environment (including each device and environmental property) as a set of objects consisting of properties and methods in order to detect feature interactions. Within our model, we formalize device interaction and environment interaction. The device interaction is in direct conflict among features of the same appliance device. This interaction arises when multiple integrated services simultaneously trigger methods that update an appliance property in different ways. The environment interaction is in indirect conflict among different appliances via the HNS environment. The interaction arises when multiple integrated services simultaneously trigger different appliances so that they try to perform an inconsistent update of an environment property.

In order to detect such interactions, we defined the HNS integrated services as a set of the relations between properties (appliance and environment) and methods of appliances. We actually created the program adopting our algorithm to detect feature interactions with practical integrated service scenarios.

5.2. Future Research

The proposed HNS-SOA framework is not perfectly superior to the conventional one. As possible drawbacks in the fully-distributed control of the appliances with SOA, the following issues are currently anticipated:

Cost of Appliances: Each appliance must be intelligent enough to satisfy Condition C2 (see Section 2.3), in order to realize the service layer. This makes the cost of appliances more expensive than the conventional ones.

Communication Overhead: It is expected that the communication overhead required

for the service orchestration cannot be ignored. Hence, when applying to the integrated services that require *hard-realtime* response, we need careful consideration.

Global Management: Since the service control is fully distributed, it is hard to manage all the appliance at once. Hence, managing the appliances all at onve requires more sophisticated mechanisms for the detection of faulty appliances and the application of global security policy, etc [47].

As to the cost of the appliances, as we explained in Section 2.3, some commercial products already exists involving Web applications so that the users can configure and control the product from PCs through a Web interface (e.g.,[40, 48]). Therefore, creating the appliance which filled the required functions for our proposal is easy.

Although HNS integrated services which need a hard-realtime response do not require communication overhead presently, this overhead should be taken into consideration regarding the practical service scenarios which need it, and the method of realization in our future research.

Global management is the most significant issue in the HNS-SOA. We have to correspond also about the appliance management of those other than integrated services, such failure detection of appliances, user authentication management and connection management with an external network, etc. The global management is required also in feature interactions detection, for practical implementation, especially in SOA. In conventional HNS-SCA, since a centralized server exists, interactions are detectable by its server. However, in HNS-SOA without a centralized server, the independent services to detect interactions are required.

In this dissertation, we proposed interaction detection by static analysis, which is the detecting approach at the time of the integrated service design. As for the interaction detection, a run-time interaction detection which is the detecting approach at the

time of the integrated service operation, also exists.

In the run-time interaction detection of the integrated services, a framework which supervises execution of each appliance is needed. In order to perform the run-time detection in HNS-SOA, we consider adding a function to report the service operation to the service layer of each appliance as a new implementation template.

In this dissertation, we developed the prototype of HNS-SOA. We are planning to add static / dynamic detection of the service interaction to our prototype, and to advance research on developing more a practical and complicated HNS application.

References

[1] D.Amyot, L.Charfi, N.Gorse et al. “Feature Description and Feature Interaction Analysis with Use Case Maps and LOTOS” *Sixth International Workshop on Feature Interactions in Telecommunications and Software Systems FIW ’00. Glasgow, Scotland*, 2000.

[2] D.Amyot, L.Logrippo (Eds.), “Feature Interactions in Telecommunications and Software Systems VII,” *IOS Press*,Amsterdam, 2003.

[3] Bluetooth, `https://www.bluetooth.org/`

[4] R.J.A.Buhr, “Use Case Maps as Architectural Entities for Complex Systems,” *IEEE Transactions on Software Engineering, Special Issue on Scenario Management*, Volume 24, Issue 12, 1998, pp.1131-1155.

[5] T.Bolognesi, E.Brinksma. “Introduction to the ISO Specification Language LOTOS” *Computer Networks and ISDN Systems* , 14, 1986, pp.25-29.

[6] L. du Bousquet, “Feature Interaction Detection using Testing and Model-checking - Experience Report” *World Congress in Formal Methods*, Toulouse, France, 1999.

[7] Business Process Execution Language for Web Services, Version 1.1,

http://www-106.ibm.com/developerworks/library/ws-bpel/
.

[8] CEBus, SmartHome Forum, http://www.smarthomeforum.com/start/cebus.asp

[9] B.G. Celler, W. Earnshaw, E.D. Ilsar, L. Betbeder-Matibet, M.F. Harris, R. Clark, T. Hesketh, and N.H. Lovell, "Remote monitoring of health status of the elderly at home. A multidisciplinary project on aging at the University of New South Wales," *Int. J. Bio-Med. Comp.*, vol. 40, pp. 147-155, 1995.

[10] E. Cerami, "Web Services Essentials – First Edition," *O'Reilly & Associates Inc.*, United Stated of America, 2002.

[11] Digital Living Network Alliance, http://www.dlna.org/

[12] ECHONET Consortium, http://www.echonet.gr.jp/

[13] M. Fowler, K. Scott, "Uml Distilled: A Brief Guide to the Standard Object Modeling Language," *Addison-Wesley, Boston*, 1999.

[14] M. van Gils, J. Parkka, R. Lappalainen, A. Ahonen, A. Maukonen, T. Tuomisto, J. Lotjonen, L. Cluitmans, and I. Korhonen, "Feasibility and user acceptance of a personal weight management system based on ubiquitous computing," *Proc. 23rd Annu. Int. Conf. IEEE Engineering in Medicine and Biology Society*, Istanbul, Turkey, Oct. 25-28, 2001 (CD-ROM, paper #581).

[15] 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.

[16] 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.

[17] S. Hariri, and C. S. Raghavendra, “SYREL:A Symbolic Reliability Algorithm Based on Path and Cutset Methods,” *IEEE Transactions on Computers*, October, pp.1224-1232, 1987.

[18] H. He, “What is Service-Oriented Architecture?,” `http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html`

[19] Hitachi Home & Life Solutions inc., “horaso network,” `http://www.horaso.com/`

[20] Home Audio/Video Interoperability, `http://www.havi.org/`

[21] HomePlug Powerline Alliance, `http://www.homeplug.com/`

[22] iReady, `http://www.sharp.co.jp/corporate/news/031217-2.html`

[23] Jini - `http://www.jini.org/`

[24] D.O.Keck, P.J.Kuehn, “The feature and service interaction problem in telecommunications systems: A survey,” *IEEE Transactions on Software Engineering* Vol. 24, No.10, pp.779-796, 1998.

[25] M. Kolberg, E. H. Magill, and M. Wilson, “Compatibility issues between services supporting networked appliances”, *IEEE Communications Magazine*, vol. 41, no. 11, pp. 136-147, Nov. 2003.

[26] I. Korhonen, J. Parkka, M. van Gils, “Health monitoring in the home of the future,” *IEEE Eng Med Biol Mag.* Vol. 22(3), pp.66-73, 2003.

[27] I. Korhonen, T. Iivainen, R. Lappalainen, T. Tuomisto, T. Koobi, V. Pentikainen, M. Tuomisto, and V. Turjanmaa, "TERVA: System for long-term monitoring of wellness at home," *Telemed. J. e-Health*, vol. 7, no. 1, pp. 61-72, 2001.

[28] LG Electronics, "Home Network" `http://www.lge.com/products/homenetwork/homenetwork.jsp`

[29] S. W. Loke, "Service-Oriented Device Echology Workflows," *Proc. of 1st Int'l Conf. on Service-Oriented Computing (ICSOC2003)*, LNCS2910, pp.559-574, Dec. 2003.

[30] J. Lotjonen, I. Korhonen, K. Hirvonen, S. Eskelinen, M. Myllymaki, and M. Partinen, "Automatic sleep/wake and nap analysis with a new wrist worn online activity monitoring device Vivago WristCare," *Sleep*, vol 26, no. 1, pp. 86-90, 2003.

[31] Matsushita Electric Industrial Co., Ltd., Kurashi net, `http://national.jp/appliance/product/kurashi-net/`

[32] A.Metzger, "Feature Interactions in Embedded Control Systems," Computer Networks, Volume 45, Issue 5, Special Issue on Directions in Feature Interaction Research, Elsevier Science, 2004, pp.625-644.

[33] R. Milner, "Communicating and Mobile Systems: the Pi-Calculus," Cambridge University Press, 1999.

[34] Mitsubishi Rayon Co., Ltd., "Home network," `http://pofeska.com/tec/homenet1/homenet1.htm`

[35] Nippon Telegraph and Telephone Corporation, "Home Service Harmony," `http://www.ntt.co.jp/news/news04/0403/040308.html`

[36] M. Ogawa, T. Tamura, and T. Togawa, "Automated acquisition system for routine, noninvasive monitoring of physiological data," *Telemed. J.*, Vol. 4, no. 2, pp.177-195, 1998.

[37] OSGi Alliance, `http://www.osgi.org/`

[38] Home Phoneline Networking Alliance, "HomePNS Specification 3.0," `http://www.homepna.org/`

[39] J. Parkka, M. van Gils, T. Tuomisto, R. Lappalainen, and I. Korhonen, "A wireless wellness monitor for personal weight management," *Proc. 2000 IEEE EMBS Int. Conf. Information Technology Applications in Biomedicine, ITAB-ITIS 2000*, Arlington, VA, pp. 83-88.

[40] PLANEX COMMUNICATIONS Inc., BRC-14V, `http://www.planex.co.jp/product/broadlanner/brc14v.shtml`

[41] S. Rhee, B-H. Yang, and H. Asada, "The Ring Sensor: A new ambulatory wearable sensor for twenty-four hour patient monitoring," *Proc. 20th Annu. Int. Conf.IEEE Engineering in Medicine and Biology Society*, Hong Kong, Oct. 1998, pp.1906-1909.

[42] Samsung, "Home Network," `http://www.samsung.com/HomeNetwork/index.htm`

[43] N. Saranummi, I. Korhonen, M. van Gils, and S. Kivisaari, "Barriers limiting the diffusion of ICT for proactive and pervasive health care," *IFMBE Proc. Medicon 2001, 9th Mediterranean Conf. Med. Biological Engineering and Computing.*, Part 1, Pula, Croatia, pp. 23-26, 2001.

[44] S. Soh, and S. Rai, “CAREL: Computer aided reliability evaluator for distributed computing networks,” *IEEE Trans. Parallel and Distributed Systems*, July, pp.199-213, 1991.

[45] Y. Tajika, D. Ajitomi, M. Nakamura, M. Minoh, “Novel Networked Appliance Architecture designed for the Integration of Distributed Monotypic Functions on a Home Network,” J*ournal of IPSJ*, Vol.44, No.9, pp.2320-2333, Sep. 2003.

[46] 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.

[47] F. Tartanoglu, V. Issarny, N. Levy, and A. Romanovsky, “Dependability in the Web Service Architecture,” *Proc. of the ICSE Workshop on Architecting Dependable Systems*, Orlando, USA, May 2002.

[48] Toshiba Corporation, “net de navi,” `http://www.rd-style.com/`

[49] T. Tsuchiya, T. Kajikawa, and T. Kikuno, “Parallelizing SDP (Sum of Disjoint Products) Algorithms for Fast Reliability Analysis,” *IEICE Transactions on Information and Systems*, Vol.E83-D, No.5, May , pp.1183-1186, 2000.

[50] UPnP Forum, `http://www.upnp.org/`

[51] W3C Web Service Activity, `http://www.w3.org/2002/ws/`

[52] W3C, “Web Services Choreography Description Language, Version 1.0,” `http://www.w3.org/TR/ws-cdl-10/`

[53] S. Weerawarana, and F. Curbera, “Business process with BPEL4WS: Understanding BPEL4WS, Part1,” `http://www-106.ibm.com/developerworks/webservices/library/ws-bpelcol1/` .

[54] M. Weiss, “Feature Interactions in Web Services,” *Proc. of Seventh Int’l. Workshop on Feature Interactions in Telecommunication Networks and Distributed Systems (FIW’03)*, pp.149-156(2003).

[55] X10, `http://www.x10pro.com/`

[56] Yoneda, T. and Ohta, T., “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.