

**Reducing Service Interaction in Home Network by Managing  
Appliance/Abstract Objects,  
Method's Execution Rule and State**

**Kimio TSUCHIKAWA Kazunori KATAYAMA  
Yoshihiro NAKAMURA and Fumihiko ITO**

NTT Access Network Service Systems Laboratories, NTT Corporation  
1-7-1 Hanabatake, Tsukuba-shi, Ibaraki, 305-0805 Japan  
Telephone +81 29 868 6135  
E-mail: {kimio, katayama, n.yoshi, f.ito}@ansl.ntt.co.jp

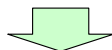
**Abstract**

We describe a resource management framework that will enable us to avoid the service interaction problem in home networks. Two kinds of the interaction (undesired interaction) have been discussed: one is a simple interaction that results when two or more services are about to operate the same networked appliance simultaneously, the other is a complicated interaction that results when there is competition for abstract objects such as environments. To deal with both types of interactions, in this approach we implement these appliance/abstract objects including their mutual influences. Moreover, to realize flexible service execution conditions that can reduce the service interactions, we propose a method's execution rule and state that enable time sharing, space sharing and/or any other item sharing of resources with the assumption of the existence of a certain negotiation process.

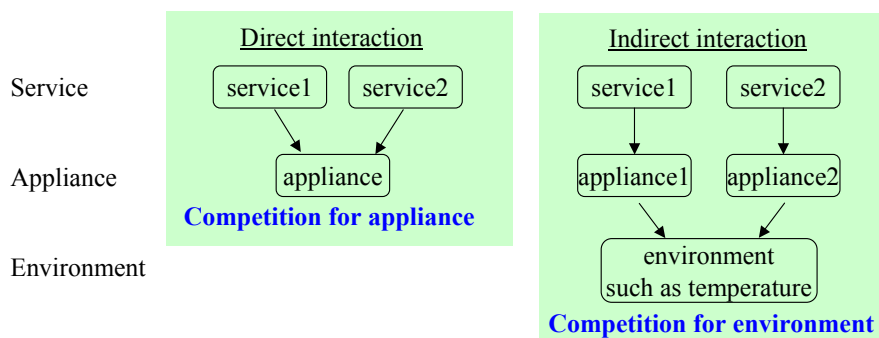
**Keywords:** home network, home service, networked appliance, abstract object, service interaction, resource management, service management

## Introduction

- Several kinds of appliances are being networked (e.g. air conditioners, washing machines, lights and sensors).
- Some kinds of home network services are being provided by each service provider (such as home security services and energy management services).
- In such situations, it is common that two or more services coexist in a home network.



Undesired interaction between the services must be detected and properly avoided.



APNOMS 2005

(2)



### 1. Introduction

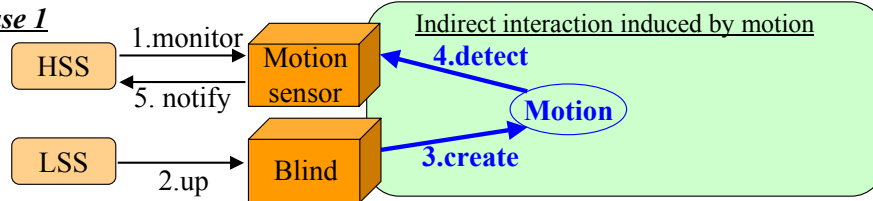
#### 1.1. Background

Recently, networked appliances that can be operated via networks (such as air conditioners, refrigerators, washing machines, lights and human detective sensors) have been appearing on the market. Home network services that provide each function by operating networked appliances, such as home security services, energy management services and entertainment services, are also commercially available.

The popularization of these services has meant that there are usually two or more services coexisting in a home network. In such situations, the interaction between these services is a very important problem. It is known that there are two kinds of service interaction [1]. One is a direct interaction that results when two or more services are about to operate the same networked appliance simultaneously. The other is an indirect interaction that results when two or more services operate different appliances but consequently control the same abstract object such as environmental feature (such as room temperature, motion and light). It is relatively easy to detect the former interaction because we only need to monitor the states of the networked appliances to detect it. It is more difficult to detect the latter interaction because the states of abstract objects such as the environment must also be monitored to detect it.

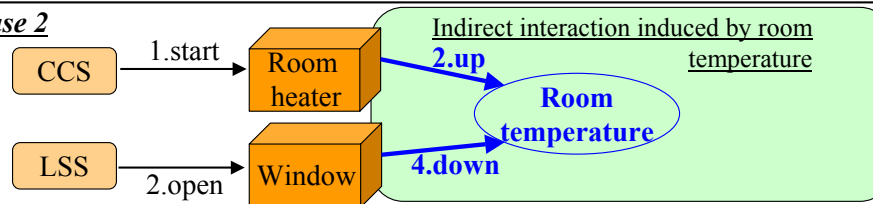
## Examples of Indirect Interaction

### case 1



- HSS : Home Security Service monitors a motion sensor that activates an alarm if motion is detected.
- LSS : Life Support Service raises a blind at 8 a.m..

### case 2



- CCS : Climate Control Service starts a room heater to heat up a room.
- LSS : Life Support Service opens a window to provide air ventilation.



NTT

APNOMS 2005

(3)



## 1.2. Examples of indirect interaction

We explain the latter indirect interaction using two examples. With case 1, shown here, we assume that two services are being provided for a certain home.

-The home security service (HSS) monitors motion in a room to detect intruders by monitoring a motion sensor. When motion is detected, it activates an alarm.

-The life support service (LSS) raises a blind to wake the user comfortably at 8a.m.

When these two services operate without any negotiation simultaneously, it is predictable that the HSS activates the alarm at 8a.m. This is because the HSS has mistakenly interpreted the motion of the blind as intruders. Needless to say, this is unpleasant for the service provider.

With case 2, we assume that two services are being provided for a certain home.

-The climate control service (CCS) starts a room heater to heat up a room.

-The life support service (LSS) opens a window for 60s every 2 hours to provide air ventilation.

This case can be also thought one of the examples of the indirect interaction for the following reason. In a cold day, the CCS heats the room temperature by a room heater, while the LSS cools it down by opening a window. Consequently, the operation of the LSS brings unpleasant effect for the CCS.

It is a general case where each service is independently provided by each service provider specializing in a certain field and each service is installed independently; therefore it is difficult for the service providers or the users to grasp operations in services correctly, and then predict what happens when these service runs simultaneously and avoid the interactions beforehand.

For the above reason, a scheme to detect and properly avoid such interaction is expected to appear so that networked appliances make human life more comfortable and safe.

## Related Work

### "Compatibility Issues between Services Supporting Networked Appliances"

Mario Kolberg et al.[1]

which defined

- environments as resources in addition to networked appliance
- influence on environment caused by networked appliance operation
- four kinds of occupation states for environment (NS,  $S_{\pm}$ ,  $S_{+}$  and  $S_{-}$ )
  - NS means that only one service or appliance can altered environment at a time.
  - $S_{\pm}$  means that a appliance changes the environmental variable in an unknown way (for example it either increases or decreases the temperature).  $S_{\pm}$  is also used for accessing environmental variables that cannot be influenced in a directed way; for instance movement cannot be increased or decreased.
  - $S_{+}$  and  $S_{-}$  means that only operations which have the same direction (increase or decrease) are allowed.
- only one solution for competition that is based on the priority

#### Remaining problem

Too-strong exclusiveness of prioritized services

(Once one service with the highest priority exclusively uses a resource, other services can not use it at all.)



APNOMS 2005

(4)



### 1.3. Related work

It is difficult to detect every kind of service interaction by monitoring only the networked appliance because some service interactions are caused by competition in environments. A research has been undertaken that focuses on this [1].

Reference [1] describes one approach where three kinds of objects (service, networked appliance and environment) are defined, and the influence on the environment caused by the execution of networked appliances is also defined to detect the indirect interaction. And four kinds of occupation state (NS,  $S_{\pm}$ ,  $S_{+}$ , and  $S_{-}$ ), which are assumed to be able to adjust to all kinds of environments, are defined to detect competition in environments.

-NS means that only one service or appliance can altered environment at a time.

- $S_{\pm}$  means that a appliance changes the environmental variable in an unknown way (for example it either increases or decreases the temperature).  $S_{\pm}$  is also used for accessing environmental variables that cannot be influenced in a directed way; for instance movement cannot be increased or decreased.

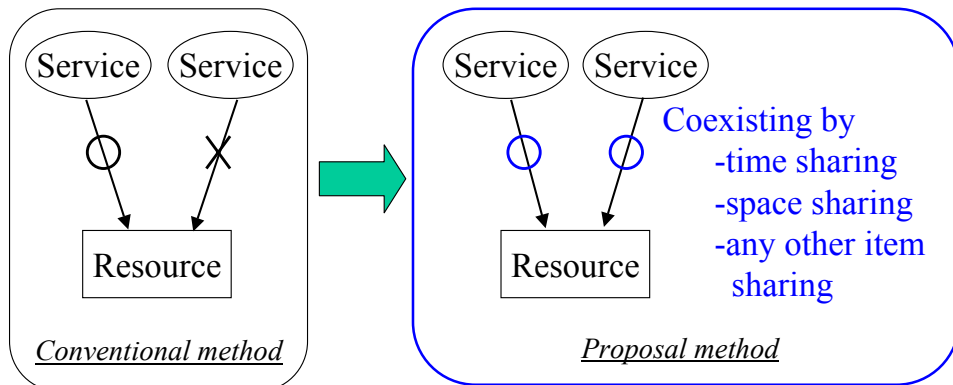
- $S_{+}$  and  $S_{-}$  means that only operations which have the same direction (increase or decrease) are allowed.

In this work, only one solution for the competition that is based on priority was also proposed, but it could happen that once one service with the highest priority exclusively uses a resource, other services can not use it at all. Such a too-strong exclusiveness is a remaining problem. With regard to the examples that we have just described, the two services can not coexist. Only one service is allowed to access the resources, and the other is disqualified.

## Purpose of This Work

### Purpose

Relaxation too-strong exclusiveness  
so that more services can provide their functions.



APNOMS 2005

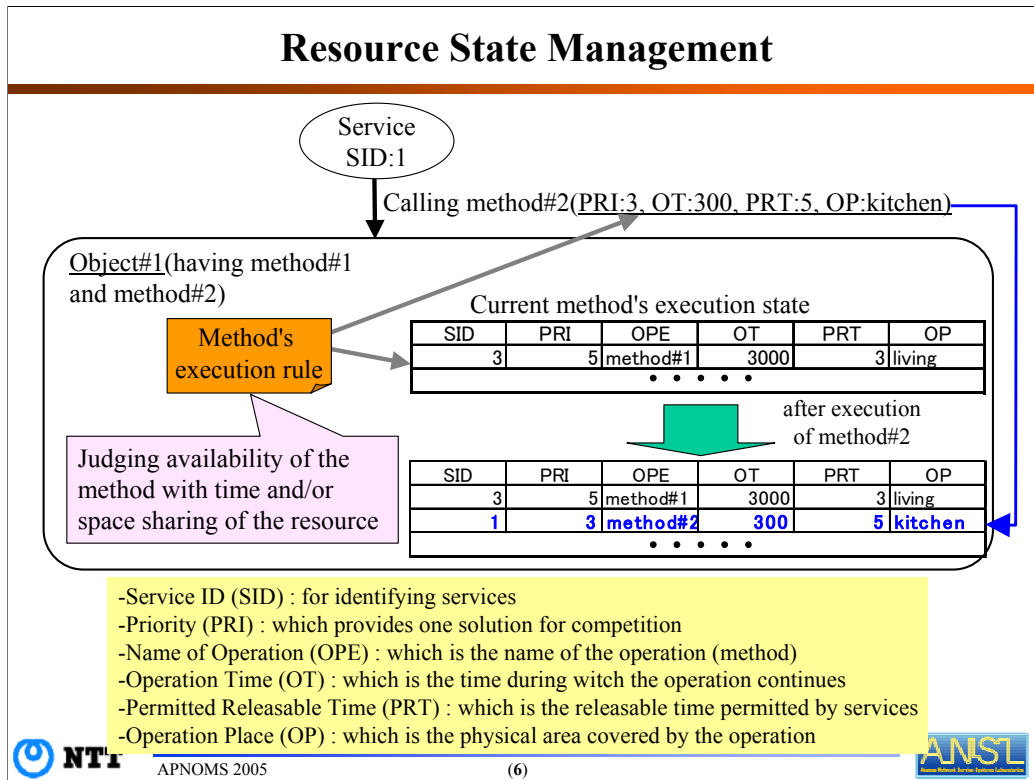
(5)



## 2. Purpose of this work

The purpose of this work is relaxing the too-strong exclusiveness such as the cited work. For this, one resource may be shared between services by time sharing, space sharing and/or sharing under other conditions. As a result, it can be expected that more services can provide their functions.

# Resource State Management



## 3. Resource management design

First of all, we mention that not only tangible networked appliances in home networks but also abstract objects such as environments are defined and managed as resources in this paper. To realize our purpose, we propose a framework where one can specify insistences and concessions for resources in detail and each resource manages these insistences and concessions. The specification is referred to as method's execution state in this paper. The method's execution state includes the following items:

- Service ID (SID) : for identifying services
- Priority (PRI) : which provides one solution for competition
- Name of Operation (OPE) : which is the name of an operation (method)
- Operation Time (OT) : which is the time during which the operation continues
- Permitted Releasable Time (PRT) : which is the releasable time permitted by services
- Operation Place (OP) : which is the physical area covered by the operation
- other conditions related to occupation and/or release permission

We believe that each resource can be shared more flexibly with services by negotiation with these information.

### 3.1. Resource state management model

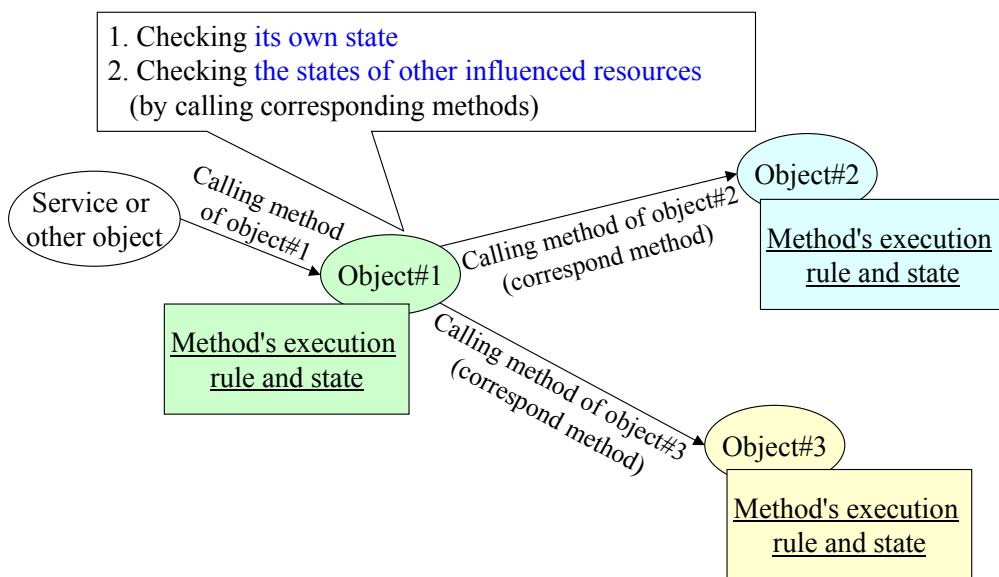
This figure depicts the resource state management model. A resource (object) has its own methods (here, method #1 and method #2). And to detect the direct interaction in the resource and reduce it properly, the resource has its own method's execution rule and manages its own method's execution state, which is shown in this table. This rule decides the operation of the resource, which is linked to the execution process of the methods. And this rule enables the resource to determine the availability of the called method by using its own method's execution state. Here, we mention that this judgment means not only alternative judgment as conventional method but also flexible judgment in term of time sharing, space sharing and/or any other item sharing the resources. For that purpose, a way of determining the availability of the called method by comparing the current execution state of the resource and the contents of the called method (such as the required service (SID), the name of the method (OPE) and the arguments (PRI, OT, PRT and OP)) is implemented as the method's execution rule.

We explain the rule in more detail with one example. A motion object (described in section (4.2)) has a lockMotion method and a createMotion method. The motion object, which has received and accepted the lockMotion method from a certain service previously, also receives the createMotion method from an other service, and can then judge that the createMotion method is acceptable if the PRT specified in the lockMotion method is bigger than the OT specified in the createMotion method by referring to the rule of the motion object. This process is implemented in the rule of the motion object as one of the judging processes. Because it is thought that these rules and states strongly depend on the characteristics of the resources, each rule and state need to be carefully designed considering the characteristics of the resource and must be implemented in each resource in order to realize the purpose.

In most cases, when a method is executed, the state table should be updated. The rule for updating the table must be individually implemented in the method of the resource, considering the characteristics of the resources.

In this way, it is realized to detect the interaction in the resource and reduce the interaction with time sharing, space sharing and/or any other item sharing.

## Interaction between Resources



### 3.2. Management interaction between resources

Next, we explain management of the indirect interaction between resources – for example, the operation of a blind that influences motion and the operation of an air conditioner that influences room temperature. In this framework, these interactions can be realized by calling the methods of the influenced resources in the implementation of the methods of the influencing resource. In this regard, we believe that what resources to call, what methods to call and how to specify arguments depend on the state of the influencing resource and the content of the called methods such as arguments. The rule for specifying these items needs to be individually implemented in the method, taking the resource characteristics into account.

Considering the former section and the above of this section, it is found that the resources need to take not only its own state but also the states of other influenced resources to determine the availability of the methods. For this, every resource needs to execute following process. When a resource (object #1) receives a request through a method (method #1), first object #1 checks the availability of method #1 by referring to its own method's execution rule and current method's execution state. Then if the result is that method #1 is available, object #1 goes through the following procedure. Object #1 checks all the influences resulting from the execution of method #1 by calling all the corresponding methods of the influenced resources and suspends itself until all replies are returned. The influenced resources check the availability of each called method by referring to their own method's execution rules and current method's execution states, and then return the results to object #1. Object #1 receives replies from the influenced resources, and if the all results of the replies show that method #1 is available, object #1 executes method #1 and properly updates the method's execution state.

## Example of Appliance Object Design

basic feature of appliance object				influence created by operation of appliance object	
	attribute	name	meaning		
		Blind object	state	state	indicates up/down state
name	meaning			object	method
method	up()		indicates blind raising operation	sound	createSound(OT:3)
				motion	createMotion(OT:3)
	down()		indicates blind lowering operation	sound	createSound(OT:3)
				motion	createMotion(OT:3)
Window object	attribute	name	meaning		
		state	indicates open/closed state		
	method	name	meaning	object	method
		open()	indicates window opening operation	sound	createSound(OT:3)
				motion	createMotion(OT:3)
				room temperature	createTemperatureChange()
close()	indicates window closing operation	sound	createSound(OT:3)		
		motion	createMotion(OT:3)		
		room temperature	deleteTemperatureChange()		
Room heater object	attribute	name	meaning		
		powerState	indicates ON/OFF state		
	method	operationMode	indicates weak/middle/strong state		
		name	meaning	object	method
		powerON()	indicates turning on a room heater operation	sound	createSound()
				room temperature	createTemperatureRise()
powerOFF()	indicates turning off a room heater operation	sound	deleteSound()		
		room temperature	deleteTemperatureFall()		
setOperationMode(mode)	indicates setting operation mode operation	none	none		
Motion sensor object	attribute	name	meaning		
		motionFlag	indicates detection state		
	method	name	meaning	object	method
	getMotion()	indicate detection state notification operation	none	none	



APNOMS 2005

(8)



### 4. Example of resource design

In this section, we provide some examples of designing resources to define and manage.

#### 4.1. Appliance object

First, we explain an appliance object, which is a model created by abstracting the resources of tangible networked appliances in home networks. We examined a blind object, a window object, a room heater object and a motion sensor object as examples of designing the appliance object. The blind object has the following attribute and methods.

- state attribute : which shows the blind state (up/down)
- up() method : which shows the blind raising operation
- down() method : which shows the blind lowering operation

Furthermore, a createMotion(OT:3) method of a motion object and a createSound(OT:3) method of a sound object are defined as the influences on other resources resulting from the execution of these methods of the blind object (up method and down method). And they will be called automatically by the blind object before the execution of these method to check the states of other influenced resources. Although we explain the abstract objects (motion object and sound object) in detail later, these mean that motion is induced for three seconds and sound is induced for three seconds, respectively. It should be noted that the former definitions are approximate to simplify them regardless of the resource state or the content of the called methods. But it is thought the approximations are reasonable, taking the operations of these appliances into account.

To aim more practicable system, it is necessary to deal with the influence created by operation of appliances, taking specific of appliances, structure of building and installation location of appliances into consideration. For this, modeling the appliance objects by vendors in view of specific of their products, a scheme to recognize and build up these influences automatically by trial operation or a history of operation of appliances, and a scheme to configure them easily by the users may be necessary.

## Example of Abstract Object Design

Motion object	attribute	name	meaning
		createMotionTimes	indicates the createMotion() method called frequency
	method	name	meaning
		createMotion()	indicates motion creating operation
		getMotion()	indicates checking presence of motion operation
method	getCreateMotionTimes()	indicates notification createMotionTimes operation	
	lockMotion()	indicates fixing motion operation	
Sound object	attribute	name	meaning
		createSoundTimes	indicates calling createSound() method frequency
	method	name	meaning
		createSound()	indicates sound creating operation
		getSound()	indicates checking presence of sound operation
		getCreateSoundTimes()	indicates notification createMotionTimes operation
		lockSound()	indicates fixing motion operation
Room temperature object	attribute	name	meaning
		currentTemperature	indicates current temperature
	method	name	meaning
		createTemperatureRise()	indicates rising temperature operation
		createTemperatureFall()	indicates lowering temperature operation
		createTemperatureChange()	indicates changing temperature operation
		setTemperature(temperature)	indicates making temperature specified temperature operation
		getTemeperature()	indicates checking current temperature operation
		lockTemperature	indicates fixing temperature operation
Intruder object	attribute	name	meaning
		intruder	indicates calling createMotion() method frequency
	method	name	meaning
		monitorIntruder()	indicates operation that notifies presence of intruders



APNOMS 2005

(9)



### 4.2. Abstract object

Next, we explain an abstract object, which means a concept or an event such as an environment operated by networked appliances (such as motion, sound, room temperature or light) and a concept, which means the intention of service providers (such as detecting intruders and the electric power consumption). Although the abstract object need to be modeled that most people can agree with to some degree so that it can be accepted generally, we think that it is possible because the abstract object is common among most people.

We examined the motion object, the sound object, the room temperature object and the intruder object as examples of designing abstract objects. The motion object that showed "motion" is created by something moving (such as a person walking or the operation of an electric fan) and is an object that is used to detect the presence of indoor motion by, for example, the home security service. The motion object has the following attribute and methods.

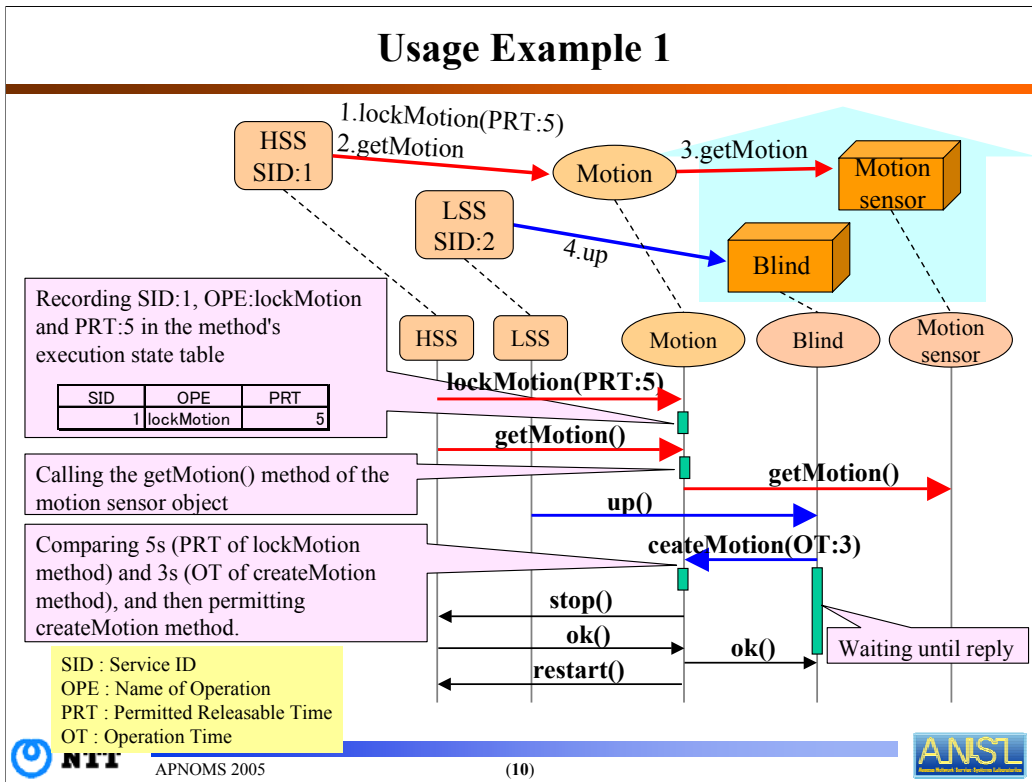
- createMotionTimes attribute : which shows the createMotion() method called frequency
- createMotion() method : which shows the motion creating operation
- getMotion() method : which indicates checking for the presence of the motion operation
- lockMotion() method : which shows the motion fixing operation

Next, we explain the room temperature object. This object has the following attribute and methods.

- currentTemperature attribute : which shows the current room temperature
- createTemperatureRise() method : which shows the temperature raising operation
- createTemperatureFall() method : which shows the temperature lowering operation
- createTemperatureChange() method : which shows the temperature changing operation
- setTemperature(temperature) method : which shows the operation that specifies the temperature
- getTemperature() method: which shows the operation used for checking the current temperature
- lockTemperature() method : which shows the fixing temperature operation

We mention that to achieve the the getMotion() of the motion object, and createTemperatureRise(), the createTemperatureFall(), the setTemperature() and the getTemperature() of the room temperature object, it is necessary to retrieve the objects and the methods that actually achieves these methods in some way and to execute them. There have already been several studies related to such abstract operation [2][3][4]. We assumed following method as existing research[4]. Each object manage concrete processes to achieve methods equipped. For example, the motion object is implemented to call the getMotion() method of the motion sensor object as a means to achieve its getMotion() method beforehand.

Finally, we mention that every object has methods with which to delete or unlock resources for the methods with which to create or lock them.



## 5. Usage Example

In this section, we explain usage examples of this method and prove how we can employ this method to solve the two indirect interactions between the services that we mentioned earlier.

### Case 1

First, we explain a solution to case 1 where the home security service (HSS) and the life support service (LSS) coexist in a room simultaneously. And to solve the case 1, the motion object, the blind object and the motion sensor object, which we have explained in section 4, are used in this method. These objects have the method's execution rules and the method's execution states designed for each resource, and operate following these rules. And we assume that no service uses resources in an initial state.

We start the explanation from the point at which the HSS (SID:1) begins operation.

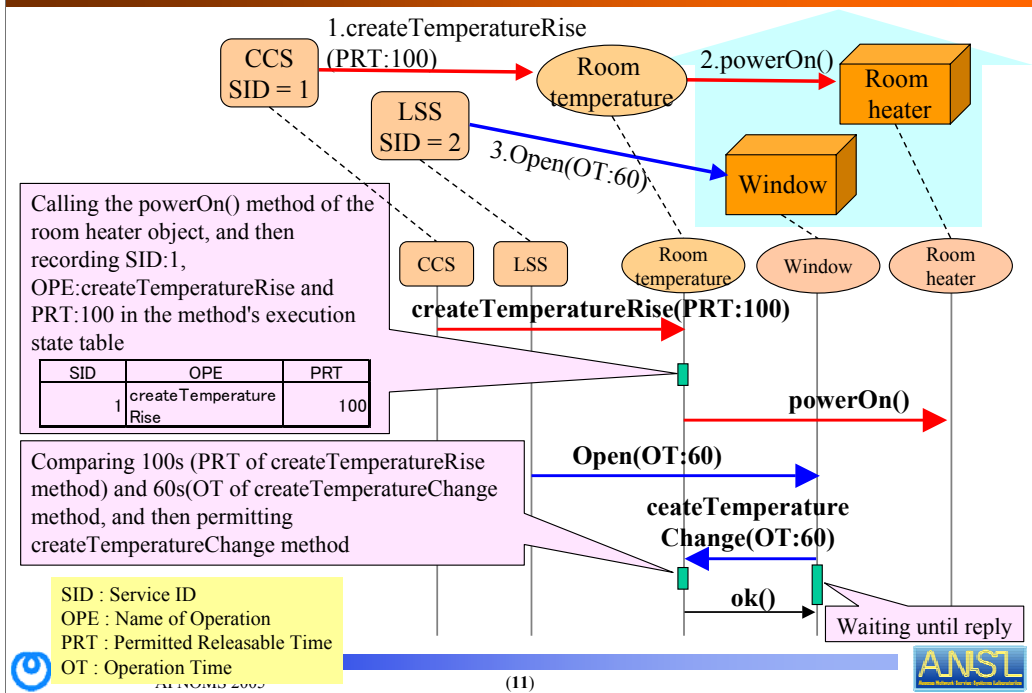
1. The HSS calls the lockMotion(PRT:5) method of the motion object, which means that a halt of monitoring motion for 5 seconds is taken the little matter and permissible for detecting intruders by the service provider.
2. The motion object first checks the availability of the lockMotion() method by referring to its method's execution rule. And then it records the SID:1, OPE:lockMotion and PRT:5 in its method's execution state by referring to its method's execution rule.
3. The HSS calls the getMotion() method of the motion object.
4. The motion object checks the availability of the getMotion() method by referring to its method's execution rule. And then it calls the getMotion() method of the motion sensor object. It is assumed that this calling process is implemented in the motion object beforehand.

Next, we explain the scenario when the LSS (SID:2) calls the up() method of the blind object.

5. The LSS calls the up() method of the blind object.
6. The blind object checks the availability of the up() method by referring to its method's execution rule, and then calls the createMotion(OT:3) method of the motion object, before the action.
7. The motion object checks the availability of the createMotion(OT:3) method by referring to its method's execution rule. Concretely it compares 5s (which is the PRT of the lockMotion method) and 3s (which is the OT of the createMotion), and then checks the availability of the createMotion(OT:3) method. Finally it permits the createMotion(OT:3) method and asks the HSS to stop its operation.
8. The HSS once stops and informs the motion object that it has stopped.
9. The motion object allows the blind object to act.
10. After 3 seconds, the motion object requires the HSS to restart.

By taking the above steps, it is possible that these two services can coexist and accomplish their purposes simultaneously.

## Usage Example 2



### Case 2

Next, we explain a solution for the second example (case 2). In this example, the climate control service (CCS) and the life support service (LSS) coexist in a room simultaneously. And to solve this case, the room temperature object, window object and room heater object, which we have explained in section 4, are used in this method. These objects have the method's execution rules and the method's execution states designed for each resource, and operate following these rules. And we assume that no service uses resources in an initial state.

We start the explanation from the point at which the CCS begins operation.

1. The CCS (SID:1) calls the createTemperatureRise(PRT:100) method of the room temperature object.
2. The room temperature object checks the availability of the createTemperatureRise(PRT:100) method by referring to its method's execution rule. After that, it calls the powerOn() method of the room heater object. It is assumed that this calling process is implemented in the room temperature object beforehand. And then, it records the SID:1, OPE:createTemperatureRise and PRT:100 in its method's execution state by referring to its method's execution rule.

Next, we explain a scenario when the LSS (SID:2) calls the open() method of the window object.

3. The LSS calls the open(OT:60) method of the window object, which means air ventilation for 60 seconds.
4. The window object checks the availability of the open(OT:60) method by referring to its method's execution rule. And then it calls the createTemperatureChange(OT:60) method of the room temperature object before action.
5. The room temperature object checks the availability of the createTemperatureChange(OT:60) method by referring to its method's execution rule. Concretely it compares 100s (which is the PRT of the createTemperatureRise method) and 60s (which is the OT of the createMotion method), and then permits the createMotion method.
6. The window object opens the window and closes the window after 60s.

By taking the above steps, it is possible that these two services can coexist and accomplish their purposes simultaneously.

## Conclusion & Future Work

### Conclusion

- We proposed reducing the (indirect) service interaction in a home network by managing
  - networked appliances / abstract objects
  - method's execution rule and state
  - inter object influence
- We provided examples of designing network appliances and abstract objects.

### Future Work

- Study of model construction methods
- Study of the other effective context for negotiation
- Study of this method functions such as authentication, security, log and accounting management
- Study of this framework software architecture (including software extension)
- Implementation, verification and evaluation of this framework



## 6. Conclusion and future work

We proposed reducing the (indirect) service interaction in a home network by managing

- networked appliances / abstract objects
- the method's execution rule and state
- inter object influence

We also provided examples of designing network appliances and abstract objects.

Future work is described below.

- Study of the model construction methods
- Study of the other effective contexts for negotiation
- Study of this method functions such as authentication, security, log and accounting management
- Study of this method software architecture (including software extension)
- Implementation, verification and evaluation of this method

### References

- [1] Mario Kolberg, Evan H. Magill, and Michael Wilson, University of Stirling, Compatibility Issues between Services Supporting Networked Appliances, IEEE Communications Magazine, pp.136-147, November 2003.
- [2] Kyeong-Deok Moon, Young-Hee Lee, and Young-Sung Son, Chae-Kyu Kim, Universal Home Network Middleware Guaranteeing Seamless Interoperability among the Heterogeneous Home Network Middleware, IEEE Transactions on Consumer Electronics, vol.49, no3, pp.546-553, August 2003.
- [3] JinNakazawa, Yoshito Tobe, and Hideyuki Tokuda, On Dynamic Service Integration in VNA Architecture, IEICE, Trans. Fundamentals, vol. E84-A, no.7 July 2001.
- [4] K. Tsuchikawa, K. Katayama, and F. Ito, "Study of Operation of Home Appliances by Abstract Service," IEICE Society Conference 2004, B-7-38, 2004.