Implementing Multi-Vendor Home Network System with Vendor-Neutral Services and Dynamic Service Binding

Masahide Nakamura[1], Yusuke Fukuoka[2], Hiroshi Igaki[1] and Ken-ichi Matsumoto[2]
[1] *Graduate School of Engineering, Kobe University*
1-1 Rokkodai-cho, Nada, Kobe, Hyogo 657-0013, Japan
{masa-n, igaki}@cs.kobe-u.ac.jp
[2] *Graduate School of Information Science, Nara Institute of Science and Technology*
8916-5 Takayama, Ikoma, Nara 630-0101, Japan
{yuusuke-f, matumoto}@is.naist.jp

Abstract

The home network system (HNS) consists of networked household appliances, intended to provide value-added services. The conventional HNS has been built on the single-vendor system, which severely limits potential of the HNS.

To overcome the problem, this paper presents a method that constructs the HNS with multi-vendor appliances. The proposed method first defines vendor-neutral standard services, with which various HNS applications and services are developed. Then, we exploit a dynamic service binding mechanism, which binds each standard service on a vendor-specific API of an appliance during run-time. With this mechanism, common HNS applications and services can be achieved by various combinations of multi-vendor appliances. Moreover, replacing any appliance with another never affects the execution of the applications.

We have implemented the proposed method using Apache Axis Web services and Rhino JavaScript engine. The experimental evaluation showed that our implementation works well for a practical HNS with sufficiently small overhead.

1. Introduction

Research and development of *home network systems* (HNS, for short) are recently a hot topic in the area of ubiquitous/pervasive computing. In the HNS, general household appliances such as TVs, DVD players, lights, air-conditioners, refrigerators, ventilators, curtains and sensors, are connected to a network at home. These *networked appliances* are controlled, monitored, and even orchestrated via the network, to provide sophisticated applications and value-added services for home users [6]. Several HNS products are already on the market (e.g., [5][10][11] [18]).

Due to lack of the programmatic interoperability [8] and the static system architecture, the current HNS is the *single-vendor* system, where all appliances and applications are manufactured by the same vendor. The single-vendor system limits the end-users to choose their favorite appliances. It also makes third-party service providers difficult to join the business. These limitations become major obstacles for popularizing the HNS for general home. The next challenge for the industries is to achieve the *multi-vendor HNS*, where various HNS applications works for any combination of multi-vendor appliances. However, sharing the same API specifications among all appliance vendors is quite difficult due to technical and political reasons.

This paper presents a method that constructs the HNS with multi-vendor networked appliances. Our key idea is to introduce *standard services* in the middle of the HNS applications and the appliances. The standard service provides *vendor-neutral service interfaces* for the HNS applications in the upper layer, encapsulating vendor-specific appliance APIs. Also, the standard service implements *dynamic service binding mechanism*, which binds the standard services to the concrete appliances during runtime. The binding can be changed flexibly just by updating the binding definition. With this mechanism, common HNS applications and services can be achieved by various combinations of multi-vendor appliances. Moreover, replacing any appliance with another never affects the execution of the applications. In the proposed framework, the standard services can be implemented and provided even by third-party providers. Also the architecture accepts a wide range of multi-vendor appliances, which allows end-users to have their favorite appliances and vendors.

Based on the proposed method, we have implemented a multi-vendor HNS platform, called *Verbena*. Verbena achieves the dynamic service bindings and the vendor-neutral services, using JavaScript and Web service tech-

978-0-7695-3283-7/08 $25.00 © 2008 IEEE
DOI 10.1109/SCC.2008.148

nologies. We have deployed Verbena in the practical HNS proposed in our previous work [13][14]. The experimental evaluation showed that the proposed method works well for a practical HNS with sufficiently small overhead.

2. Preliminaries

2.1. Home Network System

As illustrated in Figure 1, a HNS consists of *networked home appliances* and a *home server*, connected to LAN at home. Each appliance typically has a set of *control APIs*, with which the user or external software agents can control the appliance. The *home server* works as an application server, which manages various *value-added services*. It also plays a role of the gateway to the external network. Every HNS service is implemented as a software application (we call *HNS application*) that invokes the appliance APIs according to a certain service logic. The following shows a typical example of the HNS applications.

Theater Service: Orchestrating a light, a curtain, and a TV, this service allows the user to watch the TV in a theater-like setting. When the user requests the service, the light becomes dark, the curtain is closed, and the TV is turned on in the theater-surround setting, automatically.

As for the APIs, there are several granularity levels. For instance, APIs at the *network layer* may manage the network capability of the appliance, including address setup, message and signal formats, protocols. For this, many *HNS protocols* are being standardized (e.g., DLNA[2], UPnP[19], ECHONET[3], X10, HomePlug). Such APIs are at quite low level that we don't focus in this paper.

On the other hand, APIs at the *application layer (or higher)* provide easy access to logical features of the appliance, encapsulating the underlying middleware or network protocol. For instance, APIs for a light may involve `on()`, `off()`, and `setBrightness(int level)`. Such APIs are often developped by individual appliance vendors, and are deployed in a service-oriented way so that the external applications can easily use the appliance features. The well-known OSGi framework [16] is for implementing such service-level APIs. There also exists research work applying Web services to deploying the APIs [9][13].

In this paper, we refer the APIs to the ones at the application layer, not the ones at the network layer.

2.2. Limitations in Single-Vendor HNS

Due to lack of *programmatic interoperability* [8] (i.e., interoperability at the application level), most of the current

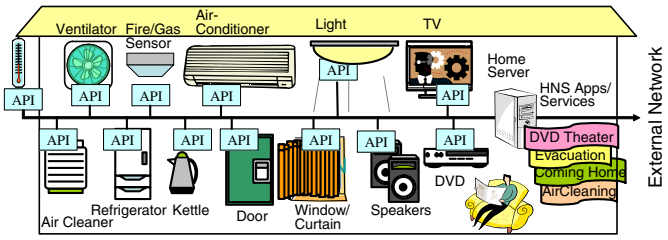


Figure 1. Example of home network system

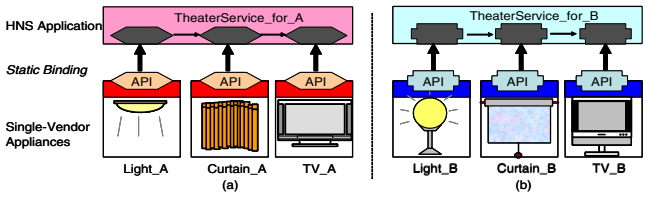


Figure 2. Conventional HNS architecture

HNS (e.g., [5] [10] [11] [18]) are comprised of the *single-vendor* system. When appliances and their APIs are developed by a vendor, HNS applications that uses the APIs are developed also by the *same* vendor. [1]

Figure 2(a) shows a typical architecture of the conventional HNS, where the **Theater Service** is implemented with appliances from Vendor A. In the figure, it is seen that `Theater_Service_A` operates `Light_A`, `Curtain_A`, `TV_A` in this order via APIs, to achieve the service requirement. Figure 2(b) shows the same service implemented by different Vendor B. Note that the implementation of `Theater_Service_B` is completely different from that of `Theater_Service_A`, since the specification of the APIs are different between Vendors A and B. Thus, it is basically impossible to mix different vendor's appliances without re-implementing the HNS applications. Even among the single-vendor appliances, the version of the APIs must be managed carefully, since any changes in APIs results in malfunction of the existing applications.

The current HNS based on the single-vendor system poses significant limitations in popularizing HNS products to the general home. From the technical viewpoint, it is difficult to replace the existing appliances by the ones with different API specifications. Also, combinations of compatible appliances are quite limited, because of the version consistency problem. From the business viewpoint, it is hard for third-party service providers to participate in the HNS business. From the end-user's viewpoint, there is little room for users to select appliances and vendors of their own choice.

[1] Some HNSs allow to use a few compatible appliances (hardware) developed by other vendors. However, controllers (including APIs) attached to the hardware are single-vendor products.

2.3. Assumptions

A straight-forward solution to achieve the multi-vendor HNS is to *standardize* the API specification shared by all vendors. Unfortunately however, this approach is not realistic due to both political and technical reasons. Every vendor in the alliance wants to take the initiative to determine the specification, so as to make full use of the own technologies. Also, a strict specification at the application level would make the appliances *uniform*, which makes it difficult for every vendor to differentiate the own product from other vendor's products in the market. Thus, it is quite hard to reach an ideal specification, where all vendors are well convinced.

Based on the discussion, we impose the following assumptions to make our problem setting clearer.

Assumption A1: For every appliance, the vendor of the appliance determines API specifications, arbitrarily.

Assumption A2: The API is implemented by the appliance vendor, and is bundled with the appliance.

Assumption A3: The API specification is opened to the consumers, including the third-party service provider.

3. Proposed Method

3.1. Requirements

Our goal is to establish a solid framework for achieving the multi-vendor HNS, where all of the following requirements are satisfied.

Requirement R1: The framework must enable the HNS to accept a wide range of multi-vendor appliances.

Requirement R2: The framework must allow every appliance to be replaced with another, without reconstructing HNS applications or re-deploying HNS platforms.

Requirement R3: The framework must allow third-party service providers to develop HNS applications and platforms.

Requirement R1 allows the HNS to have flexible combinations of multi-vendor appliances. In general, combinations of favorite appliances and vendors vary from house to house. Therefore, achieving Requirement R1 significantly enhances applicability of the HNS to various homes. Requirement R2 says that the framework should be tolerant for dynamic replacements of appliances. Every appliance has a possibility to be replaced as the consumer buys new one. Moreover, some mobile appliances could be connected

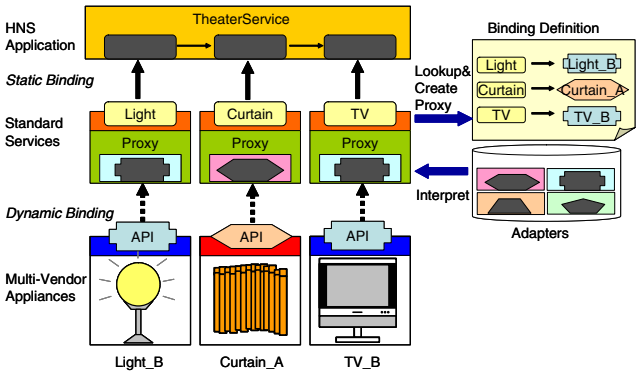


Figure 3. Proposed HNS architecture

or disconnected to the HNS, dynamically. Requirement R3 makes competitive third-parties in the HNS business, which would lead to many new services, quality improvement and cost reduction.

3.2. Overview

The problem in the conventional HNS architecture (see Figure 2) is that the HNS applications and appliances are statically coupled with vendor-specific APIs. In order to achieve the multi-vendor HNS, it is therefore essential to *weaken the coupling* between applications and appliances.

Our key idea is to exploit an additional service layer, called *standard service*, in the middle of applications and every appliance. Figure 3 shows the overview of the proposed architecture, where the Theater Service is implemented by multi-vendor appliances. Our standard services are implemented based on the following design thought:

(A) Vendor-Neutral Service Interfaces

According to Assumption A1, specifications of appliance APIs vary from vendor to vendor. To achieve Requirement R1 for such heterogeneous APIs, every standard service should provide a *vender-neutral service interface* for the HNS applications, with encapsulating the underlying vendor-specific APIs. We assume that the service interfaces are determined arbitrarily by the service provider (but not appliance vendors).

(B) Dynamic Service Binding Mechanism

Since every standard service is an abstract model of an appliance, we then have to *bind* the service to the concrete appliance actually deployed in the HNS. For this, we need to translate the vendor-neutral service interfaces into the vendor-specific appliance APIs. According to Assumptions A2 and A3, the service provider can prepare an *adapter* for every appliance, which describes how to invoke the API for the appliance. As shown in Figure 3, each standard service

Table 1. Responsible stakeholders

Components	Conventional	Proposed
Appliances	Appliance Vendor	Appliance Vendor
Appliance APIs	Appliance Vendor	Appliance Vendor
HNS Applications	Appliance Vendor	Service Provider
Standard Services	————	Service Provider
Adapters	————	Service Provider
Binding Definitions	————	Home Users

implements a *dynamic proxy*, which interprets an appropriate adapter *during runtime*, according to the user-defined binding definition. Owing to such dynamic service binding, Requirement R2 will be achieved.

Note, in the proposed architecture, that the standard services and the HNS applications can be developed by any service provider, which can be independent of specific appliances or vendors. Thus, Requirement R3 can be achieved. Table 1 compares the previous and the proposed architectures, with respect to who is responsible for each component of the HNS.

3.3. Vendor-Neutral Service Interfaces

Designing good interfaces for the standard services is important for the HNS service provider. We present a guideline on how to determine the vender-neutral service interfaces. The proposed guideline consists of two steps.

3.3.1 Step 1: Determining Target Features

Not only the API specifications, but also the *features* of the appliances varies from vendor to vendor. Therefore, for each kind of appliance, we first determine *target features*, which should be provided by the standard service. For this, we propose to choose features satisfying the following two conditions: *(a) commonly equipped by almost all vendor's appliances* and *(b) frequently used in the daily life*.

For example, Table 2 represents features of digital TV products from five major Japanese vendors. Each entry shows whether the product has the feature (✓) or not (-). From this table, we can see that the following ten features are commonly included by all vendors' products: Power On/Off, Change Sound Volume, Mute, Switch Sub-channel, Change Channel, Display Channel, Select Input, Off Timer, Channel Configuration, Picture Configuration. Any other features are equipped by only specific vendors, so choosing them as the target features yields the vendor-dependent problems in the standard service.

Among the ten features, Channel Configuration and Picture Configuration are not frequently used, once the TV has been successfully configured on its deployment. Hence, there is little benefit to exhibit them for the HNS applications. As a result, we choose the following eight features as the target features in this example: Power On/Off, Change Sound Volume, Mute, Switch Sub-channel, Change Channel, Display Channel, Select Input, Off Timer.

Table 2. Features of digital TVs from different vendors (partly shown)

Features	VendorA	VendorB	VendorC	VendorD	VendorE
Power On/Off	✓	✓	✓	✓	✓
Change Sound Volume	✓	✓	✓	✓	✓
Mute	✓	✓	✓	✓	✓
Switch Sub-Channel	✓	✓	✓	✓	✓
Change Channel	✓	✓	✓	✓	✓
Display Channel	✓	✓	✓	✓	✓
Select Input	✓	✓	✓	✓	✓
Broadcast Satellite Tuner	-	✓	-	-	✓
Off Timer	✓	✓	✓	✓	✓
On Timer	✓	-	-	-	-
Power Energy Saving	-	✓	-	✓	-
Sensor Energy Saving	-	✓	-	-	-
Subtitle	✓	✓	-	-	✓
TV Program	-	✓	-	-	✓
Program Info.	-	✓	-	-	✓
Channel Configuration	✓	✓	✓	✓	✓
Picture Configuration	✓	✓	✓	✓	✓
Clock Configuration	✓	✓	-	-	-

3.3.2 Step 2: Defining Service Interface

We then define service interfaces to operate the target features determined in Step 1. Based on the characteristic of each operation, we define a *method signature*. A reasonable way for the definition is as follows. For every fixed operation of the feature, we define a method that returns an error code. For every operation that requires explicit input values or the one that switches the mode with several values, we define a method taking parameters. Note that the semantics of the parameter values should be determined carefully in a vendor-neutral manner. It is also convenient to define interface for obtaining appliance status (`getStatus()`).

Figure 4 shows Java-like pseudo code defining the interface for a standard TV service. The method signatures have been derived from the eight target features in the Digital TV example. Each method corresponds to an operation of a target feature. Some methods takes parameters, whose semantics are written in the adjunct comment lines.

Note that the parameter semantics are also vendor-neutral. For instance, `changeVolume()` takes a parameter `vol`, ranging from 0 to 100, to specify the desired sound volume as the *percentage* to the maximum sound level. Generally, the definition of sound level varies among different vendors. For instance, let TV_A and TV_B be TVs from different vendors, whose maximum sound levels are 30 and 50, respectively. The specification of `changeVolume()` works well for both TV_A and TV_B. When TV_A is deployed in the HNS, invocation of `changeVolume(20)` is interpreted as "set sound level of TV_A to be 6 (=30*20%)". When TV_B is deployed, it is interpreted as "set sound level

```
interface TVServiceIF {
    /* Each method returns error code (0 for success) */
    int on();
    int off();
    int changeVolume(int vol); // vol: 0..100 (in %)
    int mute();
    int unmute();
    int subChannel(int mode);  /* mode: 0(Main),1(Sub)
                                  2(Main+Sub)*/
    int changeChannel(int ch); // ch: (Chanel Number)
    int displayChannel();
    int unDisplayChannel();
    int selectInput(int line); // line:(Line Number)
    int setOffTimer(int hour); // hour: Hour to shutdown
    int cancelOffTimer();
    TVStatus getStatus();      // returns TV status
}
```

Figure 4. Interface for standard TV service

of TV_B to be 10 (=50*20%)". Such translation of the parameter semantics will be actually performed in the *service adaptor*, which is introduced in the next section.

3.4. Dynamic Service Binding Mechanism

In order for HNS application to operate the multi-vendor appliances via the standard services, we implement the *dynamic service binding mechanism* in the standard service. To do this, we introduce the following three components.

3.4.1 Binding Definition

The *binding definition* defines a mapping from each standard service to a concrete appliance actually deployed in the HNS. In general, different houses can have different combinations of multi-vendor appliances. So, we suppose that the binding definition is generated for each home and is saved as an external file of the HNS platform. The file is updated when any appliance is installed or uninstalled.

Figure 5 shows an example of the binding definition for an instance of HNS. The standard services for the light, the curtain, and the TV are respectively mapped to `Light_B`, `Curtain_A` and `TV_B`, as shown in Figure 3.

3.4.2 Service Adapter

The service adapter describes how to invoke the appliance APIs when each vendor-neutral method is called. The adapter is assumed to be prepared by the service provider for every multi-vendor appliance. According to Assumption A3, the service provider knows the specification of the appliance APIs. Hence, the provider can implement the logic of each service interface by using the appliance APIs. Due to the derivation method of the service interfaces (see Section 3.3), for each method in the standard service, there must exist an appropriate API of every concrete appliance.

```
binddef NakamuraHNS {
    LightService -> Light_B;
    CurtainService -> Curtain_A;
    TVService -> TV_B;
    :
}
```

Figure 5. Example of binding definition

```
adapter TVAdapter_B implements TVServiceIF {
    TVAPI_B tvB = new TVAPI_B(); //use APIs for TV_B
    int on() {
        ret = tvB.setPower(true);
        return ret;
    }
    int off() {
        ret = tvB.setPower(false);
        return ret;
    }
    int changeVolume(int vol) {
        int maxLevel = 50;
        //Translate parameter semantics
        int v = maxLevel * vol * 0.01;
        ret = tvB.setVolume(v);
        return ret;
    }
    ...
}
```

Figure 6. Example of service adapter

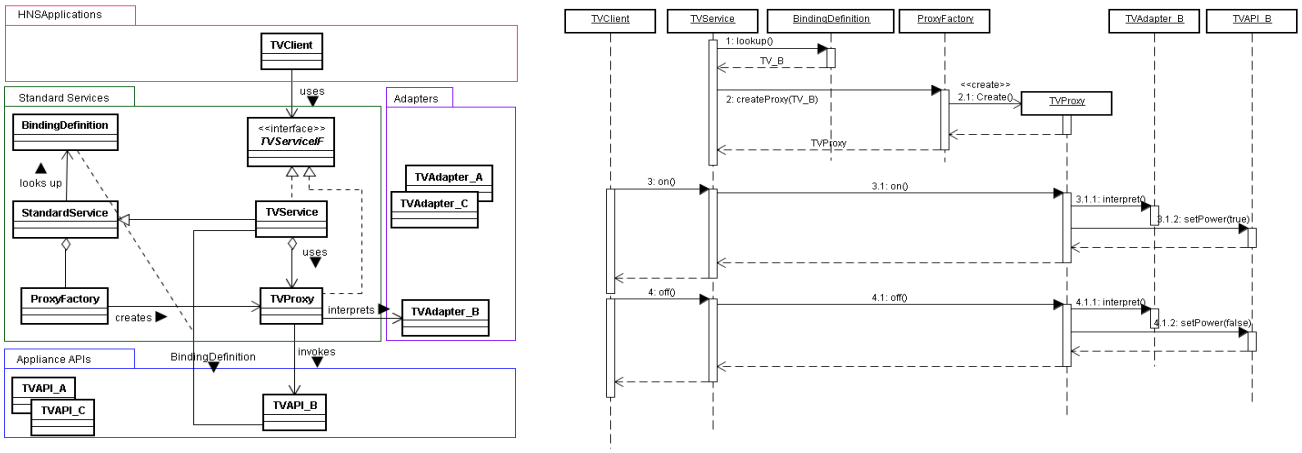
Figure 6 shows a Java-like pseudo code for the service adapter, which adapts `TV_B` to the standard TV service interface (see Figure 4). For instance, the vender-neutral methods `on()` and `off()` are implemented by the vendor-specific API `setPower()` of `TV_B`. As for `changeVolume()`, the parameter value is also adapted for `setVolume()` API of `TV_B`, based on its semantics discussed in Section 3.3.2.

The proposed framework is able to accept any appliance of any vendor, as long as the provider can prepare an appropriate adapter for the appliance. Thus, Requirement R1 is satisfied.

3.4.3 Dynamic Proxy

The dynamic proxy performs the dynamic binding of the standard service and the appliance API. When the standard service is invoked, the service creates the dynamic proxy for the target appliance, based on the binding definition. When a method of the standard service is executed, the proxy first interprets the appropriate service adapter, and then executes the appliance APIs according to the logic described in the adapter.

The dynamic proxy is created during runtime based on the binding definition. Hence, even if any appliance in the HNS is replaced, neither the HNS applications or standard services require re-construction or re-deployment. All what we need is just to update the binding definition. This achieves Requirement R2.



(a) class diagram

(b) sequence diagram

Figure 7. UML diagrams for proposed method

3.5. Design of Framework

Figure 7 depicts a UML class diagram and a sequence diagram for the proposed framework, where `TV_B` is dynamically bound to `TVService`. From the class diagram in Figure 7(a), it can be seen how `TVClient` of the HNS application executes the appliance API `TVAPI_B`, through a standard service `TVService`. The relation between `TVService` and `TVAPI_B` is defined by `BindingDefinition`. Also, the proxy `TVProxy` interprets the service adapter `TVAdapter_B`.

The sequence diagram in Figure 7(b) shows how the components communicate with each other. It is seen that the invocation of `on()` (or `off()`) is dynamically translated to `TVAPI_B.setPower(true)` (or `TVAPI_B.setPower(false)`, respectively).

4. Verbena: Platform for Multi-Vendor HNS

Based on the proposed method, we have implemented a HNS platform, called *Verbena*.

4.1. Implementation

Technologies used for implementing the system components are summarized as follows:

Standard service:	Java J2SE 5.0
Service adapter:	JavaScript
Dynamic proxy:	Mozilla Rhino 1.6 R7
Service platform:	Apache Axis 1.3 Web service

What specifically interesting in Verbena is that we adopted *Javascript* for describing the service adapters. The expressive power of JavaScript may help not only describe the invocation of the appliance APIs, but also construct sophisticated logic to improve the quality and reliability of the adapter. This contributes to covering a wide range of multi-vendor appliances, as stated in Requirement R1. Each adapter is interpreted by the dynamic proxy with a JavaScript engine, Rhino [12] during runtime. Figure 8 shows an example adapter, which binds `TVService` to the appliance API of `TV_B` (see also Figure 6).

```
TVService = new function() {
    // Logic implementing TVService.on()
    this.on = function() {
        // Invoking appliance API for TV_B.
        var ret = new Packages.TV_B().setPower(true);
        // Translating return value.
        if (ret == true) {
            return new Packages.java.lang.Integer(0);
        } else {
            return new Packages.java.lang.Integer(1);
        }
    };
    ...
}
```

Figure 8. Adapter for `TV_B` in JavaScript

To derive the vendor-neutral service interfaces for Verbena, we have investigated various appliance products available in the market. Currently, Verbena has standard services consisting 17 classes of appliances, including TV, air-conditioner, fan, ventilator, air-cleaner, blind, curtain, sound system, DVD/HDD recorder, light, gas valve, door, emergency switch, fire alarm, telephone, fax, TV telephone. To maximize the programmatic interoperability, we have deployed the standard services as Web services with Apache Axis.

4.2. Deploying Verbena in NAIST-HNS

We have then deployed Verbena in *NAIST-HNS* [13][14], which is the existing HNS developed in our previous work.

The NAIST-HNS consists of *legacy appliances* with the conventional infra-red controls, where the infra-red operations are aggregated within coarse-grained and self-contained services. Then, these services are exhibited as Web services, which adapts the legacy appliances as networked appliances. However in the previous version, the services were statically coupled with the legacy appliances. Therefore, it was not easy to replacing every appliance with another, without reconstructing the service components.

We introduced Verbena so that it dynamically binds the standard services with the legacy appliances. We then revised all the applications and services in the NAIST-HNS so as to use standard services of Verbena. As a result, the NAIST-HNS is now able to handle not only legacy appliances, but also multi-vendor and multi-protocol appliances.

4.3. Experimental Evaluation

We have conducted an experiment to demonstrate the dynamic service binding mechanism of Verbena. We implemented a new light API operating with UPnP [7][19]. The scenario of the first experiment is that we replace a light in the NAIST-HNS (infra-red control) by the new one (UPnP), during the execution of DVD Theater Service. It was seen that the DVD Theater Service first used the old light, and then switched to the new one as the binding definition of the light service was updated. No reconstruction of the DVD Theater Service or no re-deployment of the light service were needed, which successfully achieved Requirement R2.

Next, we conducted another experiment to evaluate the overhead posed by Verbena. Running the same HNS applications under the original NAIST-HNS and the one with Verbena, we compared the response time. The experiment was performed on a PC with Core Solo U1400, 1.5GB memory, Windows XP. Table 3 summarizes the result, showing the average values of 10 trials. From the result, it can be seen that some overhead is posed by the proposed dynamic binding mechanism. However, we consider the overhead sufficiently small for general use of the HNS.

Table 3. Response time of HNS applications

HNS Applications	LeaveHome	DVDTheater	AirCleaning
# of appliance APIs executed	5	5	2
NAIST-HNS: Static Binding [ms]	9,709	17,369	6,659
Verbena: Dynamic Binding [ms]	9,916	17,522	6,709
Difference [ms]	207	153	50
Relative Overhead	2.1%	0.8%	0.7%

5. Discussion

5.1. Advantage of Proposed Method

The proposed method introduces the standard services designed extensively with the vendor-neutral service interfaces and dynamic service binding. As discussed in Section 3, Requirements R1, R2 and R3 have been well achieved. Using the proposed method, the service provider can develop and distribute common applications for various combinations of multi-vendor appliances. This significantly improves the *productivity* of new services. The point is that the way of using appliances is not very different among users, although favorite appliance vendors vary significantly. In the proposed framework, the responsibilities for the HNS can be distributed reasonably for different stakeholders, as shown in Table 1. We consider it important to reduce the complexity and improve the modularity of the HNS components.

The basic idea of the proposed method is not limited within the HNS domain only, but can be applied well to general service-oriented and component-based systems [17]. For instance, let us consider a service mash-up for buying a CD: (1) the user first searches the information of the CD with a search engine, (2) the user then orders the CD to an online store, (3) the user finally asks a credit card company to perform the payment transaction. The problem is that we want to allow every user to choose a favorite combination of the search engine, the online store, and the credit card company. The problem can be achieved using the proposed method, by dynamically binding the standard service and the concrete service chosen by the user. Theoretically, the proposed method can also be used to construct enterprise system with multi-vendor service components. We will investigate the feasibility in our future work.

5.2. Limitation

To cover as many vendors as possible, the proposed method discards the vendor-specific features in deriving service interfaces (see Section 3.3.1), which would be a limitation. However, this is an inevitable trade-off between generality and specialty of the appliance features. If one wants to use such vendor-specific features in the multi-vendor HNS, a possible solution is to prepare a method that can invoke any appliance API directly, for given API name and parameters using *reflection*. The method signature is like:

```
Object invokeAPI(String name, Object [] params);
```

However, using such methods instead of vender-neutral ones significantly decreases the *portability* of the HNS application, which should be well considered.

5.3. Related Work

It is known that the OSGi framework provides the *dynamicity* for objects including smart home appliances [1][16]. However, this dynamicity is just to allow the object to be dynamically loaded and unloaded to the system. It does not deal with dynamic binding nor adaptation of heterogeneous APIs.

It seems that the proposed method takes the *adapter pattern* [4] to bridge the interface gap between the standard service and the appliance APIs. However, our method determines the target adapter during runtime, which enables to handle even unknown adaptees (i.e., appliances).

The idea regarding a concrete appliance as an abstract service is similar to the concept of the *Service Component Architecture (SCA)* [15]. However, there exist no application of SCA to the multi-vendor HNS, as far as we know.

6. Conclusion

We have proposed a method that constructs the HNS with multi-vendor appliances. Exploiting the vendor-neutral service interface and the dynamic service binding mechanism, the proposed method allows the service provider to develop HNS applications within the multi-vendor HNS. Based on the proposed method, we have implemented a platform called Verbena. We also have shown its effectiveness through experiments.

As a future work, we are currently investigating the feasibility of the proposed method to multi-vendor enterprise systems.

Acknowledgment: This research was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Young Scientists (B) (No. 18700062) and Scientific Research (B) (No. 17300007), by JSPS and MAE under the Japan-France Integrated Action Program (SAKURA), and by Oki Electric Industry Co., Ltd.

References

[1] J. Bourcier, A. Chazalet, M. Desertot, C. Escoffier, C. Marin, "A Dynamic-SOA Home Control Gateway," *Proc. of Int'l Conf. on Service Computing (SCC'06)*, pp.18-22, Sep. 2006.

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

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

[4] E. Gamma, R. Helm, R. Johnson, J. Vlissides, "Design Pattern: Elements of Reusable Object-Oriented Software," Addison-Wesley Professional, 1994.

[5] Hitachi Home & Life Solutions inc., "HORASO Network Service", http://ns.horaso.com/index.html

[6] 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.

[7] S. Konno, "CyberLink for Java", http://www.cybergarage.org/net/upnp/java/

[8] G. Lewis, E. Morris, L. O'Brien, D. Smith, and L. Wrage "SMART: The Service-Oriented Migration and Reuse Technique", *Technical Note CMU/SEI-2005-TN-029*, Software Engineering Institute, Sep. 2005.

[9] 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.

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

[11] Matsushita Electric Works, Ltd., "Lifinity", http://biz.national.jp/Ebox/kahs/index.html

[12] Mozilla Foundation, "Rhino: JavaScript for Java", http://www.mozilla.org/rhino/

[13] M. Nakamura, A. Tanaka, H. Igaki, H. Tamada, K. Matsumoto, "Adapting Legacy Home Appliances to Home Network Systems Using Web Services," *Proc. of Int'l Conf. on Web Services (ICWS 2006)*, pp.849-858, Sep. 2006.

[14] M. Nakamura, A. Tanaka, H. Igaki, H. Tamada, K. Matsumoto, "Constructing Home Network Systems and Integrated Services Using Legacy Home Appliances and Web Services," International Journal of Web Services Research, Vol.5, No.1, pp.82–98, January 2008.

[15] Open SOA Collaboration, "Service Component Architecture" - http://www.osoa.org/display/Main/Service+Component+ Architecture+Home

[16] OSGi Alliance - http://www.osgi.org/

[17] M. P. Papazoglou, D. Georgakopoulos, "Service-Oriented Computing", *Communications of the ACM*, Vol. 46, No.10, pp.25-28, 2003.

[18] TOSHIBA, "Toshiba home network – Feminity", http://www3.toshiba.co.jp/feminity/feminity_eng/

[19] UPnP Forum - http://www.upnp.org/