International Journal of Web Services Research, 5(1), 81-97, January-March 2008

Constructing Home Network Systems and Integrated Services Using Legacy Home Appliances and Web Services

Masahide Nakamura, Kobe University, Japan

Akihiro Tanaka, Nara Institute of Science and Technology, Japan

Hiroshi Igaki, Kobe University, Japan

Haruaki Tamada, Nara Institute of Science and Technology, Japan

Ken-ichi Matsumoto, Nara Institute of Science and Technology, Japan

ABSTRACT

This article presents a framework that adapts the conventional home electric appliances with the infrared remote controls (legacy appliances) to the emerging home network system (HNS). The proposed method extensively uses the concept of service-oriented architecture to improve programmatic interoperability among multi-vendor appliances. We first prepare APIs that assist a PC to send infrared signals to the appliances. We then aggregate the APIs within self-contained service components, so that each of the components achieves a logical feature independent of device/vendor-specific operations. The service components are finally exhibited to the HNS as Web services. As a result, the legacy appliances can be used as distributed components with open interfaces. To demonstrate the effectiveness, we implement an actual HNS and integrated services with multi-vendor legacy appliances.

Keywords: *home network system; infrared control; integrated services; legacy migration; service-oriented architecture*

INTRODUCTION

The emerging technologies enable general household appliances, such as TVs, DVD players, lights, ventilators, refrigerators, air conditioners, blinds and curtains, to be connected with a local area network at home. A system consisting of such *networked appliances* is generally called a *home network system* (*HNS*, for short), which is intended to provide more convenient and comfortable living for home users. Research and development of the HNS are currently a hot topic in the area of ubiquitous/pervasive

computing. Several HNS products are already on the market, (e.g., Hitachi, 2003; Matsushita, 2005; Toshiba, 2005).

The HNS provides many applications and services. The applications typically take advantage of wide-range control and monitoring of appliances inside and outside the home. Moreover, integrating different appliances via network yields more value-added and powerful services (Kolberg, Magill, & Wilson, 2003), which we call *HNS integrated services*. For instance, integrating a TV, a DVD player, speakers, lights and a curtain would implement a HNS integrated service, say, *DVD theater service*. When a user requests the service, the lights become dark, the curtain is closed, the 5.1ch speakers are selected, the sound volume is adjusted, and the contents are played with the DVD player. Thus, the user can watch movies in a theater-like atmosphere within a single operation.

In general, each networked appliance is equipped with smart embedded devices, including a network interface, a processor and storage, in order to provide and execute the appliance features required for various HNS applications and services. As the embedded devices become more down-sized, cheaper, and more energy-saving, it is expected in the near future that every object will be networked (Geer, 2006).

However, transition to the networked appliances is gradual. Most people are still using *legacy appliances*, which are the conventional non-networked home appliances. Although it is usual to see a network and PCs at home, the networked appliances are not widely spread yet.

There are several reasons why the networked appliances are not spread yet. Firstly, the networked appliances are still quite expensive. Secondly, types of available appliances are limited (audio/visual appliances have been being networked recently, but many others are not yet). Also, due to the lack of *programmatic interoperability* (Smith & Meyers, 2005), the integration of appliances is strictly limited; especially in the multi-vendor environment the integration is quite a challenging problem. Finally, there is a major requirement that the users want to keep using the legacy appliances that they are accustomed to use. Considering these reasons, it is not easy for the general home users to renew immediately all the existing legacy appliances with the networked ones.

To cope with both the emerging HNS and the legacy appliances, this article presents a new framework that adapts the legacy appliances to the HNS. Specifically, for the legacy appliances with the conventional *infrared remote controllers* (denoted by *IrRC*), we propose a way to implement a *smart adapter* on a PC that connects the legacy appliances to the HNS. For this, we exploit the concept of the service-*oriented architecture* (SOA) (Loke, 2003; Papazoglou & Georgakopoulos, 2003), extensively.

The adaptor is based on a three-layered architecture: *IR device layer*, *service layer,* and *Web service layer*. In the IR device layer, we develop a set of APIs, called *Ir-APIs*, by which the PC can send any infrared signals to appliances. Note that the infrared signals are specific to devices and vendors. Also, executing a feature of an appliance requires the user vendor-specific operations of the IrRC. Thus, it is inconvenient for external HNS applications to use the Ir-APIs directly. Therefore, the service layer then aggregates multiple Ir-API calls within self-contained *services*, so that each of the service achieves a logical feature independent of the vendor (or device)-specific issues. Finally, the services are deployed in the HNS as *Web services* (W3C, 2002) in the Web service layer. Thus, every legacy appliance becomes a distributed component with an open interface, which can be used by various kinds of HNS applications. The users can build their own integrated services and HNS applications with the legacy appliances.

To demonstrate the effectiveness, we have implemented an actual HNS and several integrated services. As a result, it was shown that the proposed framework is well applicable to multi-vendor legacy appliances, and that practical integrated services can be created as relatively small client applications. We also

present graphical user interfaces for operating HNS integrated services.

The digest version of this article was published as a conference paper in IEEE ICWS'06 (Nakamura, Tanaka, Igaki et al., 2006). Changes were made to this version, most significantly the refinement of the architecture, the addition of the user interface implementation, and the addition of evaluation section. We believe that these new results clarify the applicability and limitations of the proposed method against a practical HNS.

PRELIMINARIES

Home Network System (HNS)

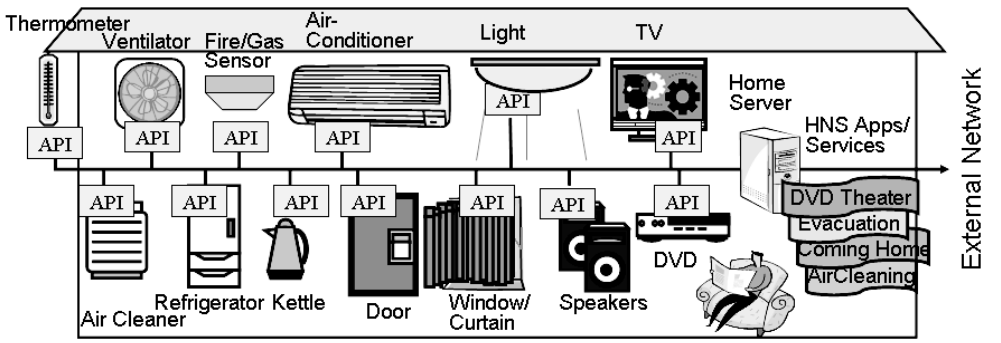
A HNS consists of one or more networked appliances connected to LAN at home. Each networked appliance has a set of *control APIs*, so that the user or software agents can control the appliance via the network. To process the API calls, each appliance generally has embedded devices including a processor and storage.

Figure 1 shows an example of HNS, which consists of various networked appliances and a *home server*. The home server typically plays a role of gateway to the external network. It also works as an application server, where the HNS applications are installed. As seen in Figure 1, every HNS integrated service is implemented as a software application that invokes the APIs according to a certain control flow. The services are supposed to be installed in the home server.

Communications among the appliances are performed based on the underlying network protocol. Currently, many standard protocols are being standardized for the networked appliances. Major protocols include DLNA (DLNA, 2006) for digital audio/video appliances and ECHONET (ECHONET, 2006) for *white* goods (e.g., refrigerator, air conditioners, and laundry machines). However, these standard protocols mainly prescribe a set of *network-layer* agreements of the appliance, such as address setup and message formats. The *programmatic interoperability* (Smith & Meyers, 2005) at the application layer or higher is beyond the scope of the protocols.

To minimize the interoperability issue, most of the current HNS (e.g., Hitachi, 2003; Matsushita, 2003; Toshiba, 2005) are comprised of the *single-vendor* appliances. Applications on the HNS are limited to the proprietary ones provided by the same vendor. Types of appliances that can be integrated are limited, too. The next challenge for the industries is to establish standards at the application layer, which allows any combination of multi-vendor appliances over different protocols.

Figure 1. An example of home network system



Service-Oriented Framework for HNS (Igaki, Nakamura, & Matsumoto, 2005)

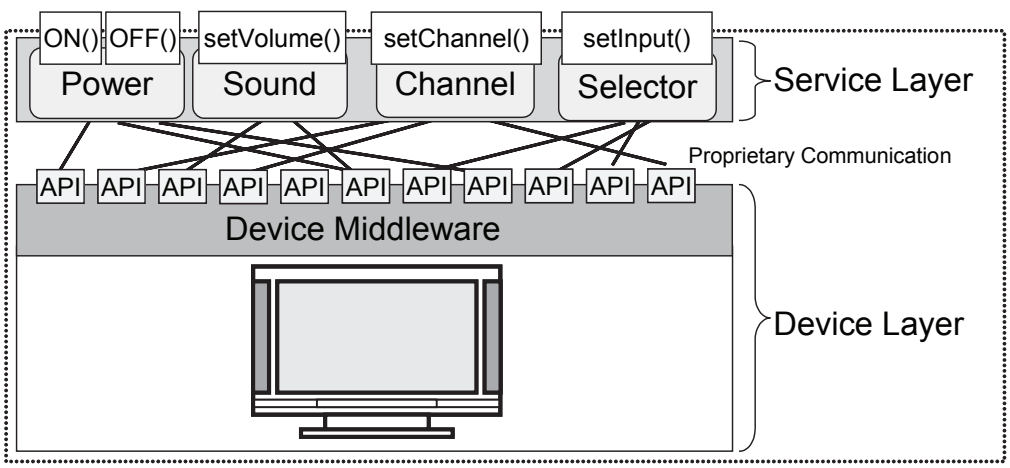
Service-oriented architecture (SOA) is a system architecture that facilitates integration of distributed heterogeneous systems (Loke, 2003; Papazoglou & Georgakopoulos, 2003). In an SOA, primary features of each system are aggregated as a set of *services*. More sophisticated systems are basically constructed by *integrating* the existing services. SOA has been extensively studied and adopted in the domain of *enterprise systems*, since SOA-based systems are quite resilient for changes and integration of business processes. *Web services* (W3C, 2002) are known as a primary means to implement SOA-based systems.

To improve the programmatic interoperability discussed above, we have presented a *service-oriented framework for HNS* in our previous work (Igaki et al., 2005), which applies the concept of SOA to the networked appliances. Figure 2 depicts an example of a networked TV, illustrating the key idea of the framework. In this framework, each networked appliance is designed based on a two-layered architecture: a *device layer* and a *service layer*. The device layer represents the hardware and the control APIs (including the middle-ware) of the networked appliance. In the figure, the APIs denote the control APIs of the networked TV. They can be invoked by external entities in accordance with the HNS protocol conformed by the TV.

On the other hand, the service layer aggregates the control APIs according to the logical features of the TV. Then, the layer exports the features to the network as the *self-contained* services with *open interface* (i.e., methods). The communication among the device and service layer is supposed to be based on the device-proprietary procedure. Thus, the proprietary API calls at the device layer are hidden from the service user. The service layer is implemented as an application, and is supposed to be installed in the storage of the networked appliance.

As seen in Figure 2, features like power, sound and channel are quite generic features that every vendor's TV is supposed to have. Thus, in this example these features are considered to be *TV services*. The methods like ON(),OFF(), setVolume(), setChannel() are open and vendor-neutral interfaces for the TV service. Invocation of these methods does not require any knowledge specific to the underlying implementation or protocol of the device layer. Unless the interface definition is changed, any modification in the service and device layers

Figure 2. Architecture of networked appliance based on SOA



does not give impact to the service users. Also, the service users can easily develop their own integrated services, by combining the method invocations of different appliances. As a result, we are able to achieve a HNS that copes with both system evolution and appliance interoperability.

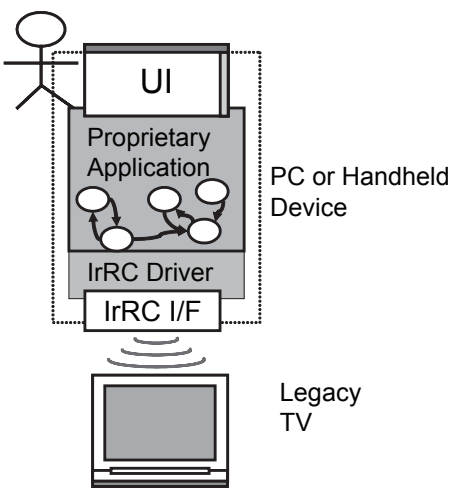
Note however that this framework presented in (Igaki et al., 2005) assumes the networked appliances only. The legacy appliances without the processor or the storage are beyond the scope. Also, the framework is not yet fully evaluated or implemented with the actual networked appliances.

Software Controller for Legacy Appliances

There have been several software applications with which the user can control legacy appliances from a PC or a handheld device such as handy phones (Kaden Control Lab, 2006; NANO Media Inc., 2005). We here call such applications *soft-controllers*.

Figure 3 shows a typical example of the soft-controllers, which is for a legacy TV with the conventional *infrared remote controller for home appliances* (denoted by *IrRC*). The user first selects and executes a control command through the user interface (UI). Then, the application sends the corresponding infrared signals to the appliance through the driver (IrRC driver) and the interface (IrRC I/F)[1].

Figure 3. Soft-controller for a legacy TV



The soft-controllers basically assume a use case that a human user controls a single appliance at a time. They are not supposed to be invoked by other applications, or to be orchestrated by other appliances via the network. Also, since the appliance and the application are tightly coupled, the same controller cannot be used directly for other appliances. Therefore, it is difficult to apply the soft-controllers directly for the purpose of adaptation of legacy appliances to the HNS.

ADAPTING LEGACY HOME APPLIANCES TO HNS

Requirements

Our goal is to adapt the legacy appliances with the IrRC to the HNS. More specifically, we try to propose a framework to implement an *adapter* for the legacy appliances, satisfying the following requirements.

- **Requirement R1:** The framework must achieve easy creation of HNS integrated services with arbitrary combinations of the legacy appliances.
- **Requirement R2:** The framework must be implemented using generic PCs and IrRC devices without special hardware.
- **Requirement R3:** The framework must be applicable to a wide range of types and vendors of the appliances.

The current HNS products do not assume integration of the legacy appliances with the HNS. Also, they have yet problems in requirements R1 and R3. Our previous framework in (Igaki et al., 2005) takes requirements R1 and R3 into account, but it cannot satisfy Requirement R2. The soft-controllers are for legacy appliances, but cannot satisfy requirement R1. Thus, the existing methods cannot be used directly to achieve our goal.

Proposed Architecture

To satisfy requirements R1 through R3, we here propose a new architecture depicted in Figure 4. As an example, we explain the architecture of an adapter for a legacy TV in the figure. The proposed architecture replaces the device layer of our former architecture (see Figure 2) with the *IR device layer* consisting of IrRC devices and the legacy appliance. Also, it puts a *Web service layer* on top of the service layer.

To implement the adapter, we first prepare a PC (or handheld device) and an IrRC device that can be connected to the PC. On the PC, we implement a set of APIs called *Ir-APIs* with which applications can send any infrared signals to the appliance. Next, for a given legacy appliance, we develop a set of services of the appliance, each of which encapsulates several Ir-API calls to achieve a self-contained and vendor-neutral feature. Finally, we export these services as Web services and deploy them in the HNS. We call the methods of the Web services *Web-APIs*.

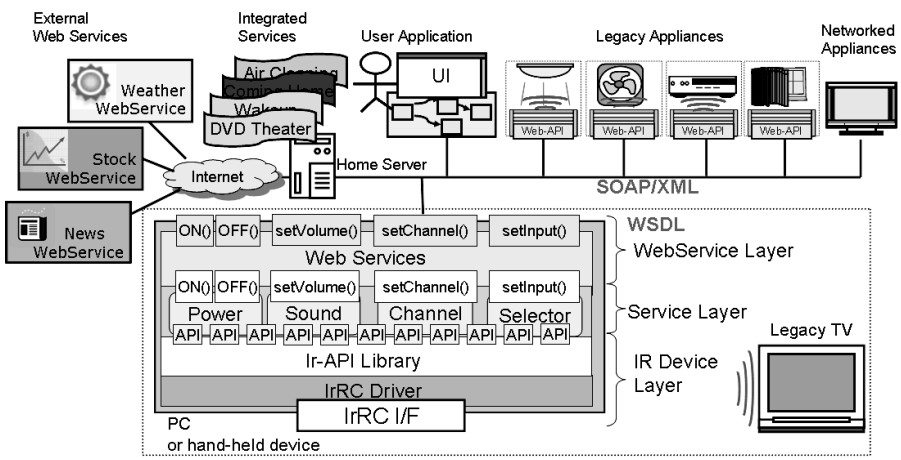
The integrated services and any user applications are implemented as client applications that invoke the Web-APIs. For this, the implementation of IrRC, which often varies among appliance types and/or vendors, is hidden within the service layer. The service users can execute various features of the appliance without device-specific knowledge. The same framework is applied to other legacy appliances. Then, the integrated service can be created by assembling Web-APIs provided multiple appliances, according to a desired control flow (i.e., *workflow*).

Taking full advantage of Web services, the proposed architecture also aims the integration of the legacy home appliances with the external Web services for the future extension (see left-side of Figure 4). Once every legacy appliance is adapted as a Web service, it is quite easy to implement services that *mash up* the appliances and various existing services available in the Internet. This fact implies that the proposed architecture has the great extendibility of HNS services.

IR Device Layer: Providing APIs for Infrared Remote Controller (Ir-APIs)

In general, a legacy appliance with an IrRC is operated as follows. When a user presses a button of the IrRC, an infrared signal corresponding to the button is issued from the controller. Upon receiving the signal, the appliance executes the corresponding feature (or operation). Thus, the communication mechanism is quite simple. However, the correspondence between the infrared signal and the feature varies among

Figure 4. Proposed architecture for adapting legacy appliances



vendors and types of the appliance. Therefore, it is convenient to have *all-purpose wrappers* for the IrRC, with which the applications in the upper layer can flexibly switch and manage such vendor (and device)-specific signals.

For this purpose, we implement Ir-APIs on the top of the Ir-RC driver. The Ir-APIs provide a set of generic interfaces with which the applications can send any infrared signals to *arbitrary* types of legacy appliances. The Ir-APIs are relatively low-level but generic APIs, which should be commonly used by *all types* of appliances. Therefore, typical Ir-APIs must include; initialize IrRC, set signal type, send signal, start sending burst signal, stop sending burst signal, sleep.

Service Layer: Aggregating Features as Vendor-Neutral Services

The granularity of Ir-APIs is so fine that one Ir-API does not necessarily correspond to a single logical feature of the target appliance. Also, each IrRC operation heavily depends on the vendor and type of the appliance. It is not a good idea to expose every Ir-API directly to the HNS, since the user has to take care of the device-specific specification of IrRC of the target appliance.

The service layer aggregates, for every logical feature of the appliance, several Ir-API calls within a *service method*. What most important here is that every service method must be *self-contained*. Specifically, we recommend that every service method *m* should satisfy the following conditions:

- **Condition S1:** *m* is always executable by itself, independent of the context of other services or appliances.
- **Condition S2:** *m* achieves by itself a consistent logical feature of the appliance.

For instance, let us consider a TV manufactured by a vendor *A*, denoted by TV_A. Suppose that TV_A has the following restriction on its implementation: "Once turned on, it does not accept any infrared signals during 2 seconds until the hardware becomes fully operational." Then, the service method ON() for TV_A should be implemented as in the Java-like pseudo code in Figure 5. The first four statements are for achieving a logical feature "turn on TV_A," satisfying condition S2. The last statement, sleep(2), is for the subsequent method invocations to achieve the condition S1. The sleep statement consumes the 2-seconds wait, so that any methods executed after ON()should not be influenced by the device-specific restriction of TV_A.

When a feature requires a sequence of *multiple* Ir-APIs, these APIs should be encapsulated within a service method. The pseudo code in Figure 6 is to set the sound volume of TV_A to a given level *x*. In general, the volume level is adjusted by a human user with a relative scale considering the current volume level. However, here the application needs to do the task. Hence, the method setVolume() first minimizes the sound volume by repeatedly sending the signal

Figure 5. Service method ON() for TVA

```
public void ON(void) {
 IrControler con = new IrController();  /*Controller and */
 IrSignal sig = new IrSignal();    /*signal objects of Ir-API*/
 sig.setSignalType(ON, TV_A);      /*set signal ON for TV_A*/
 con.sendSignal(sig);              /*send the signal*/

 sleep(2);                         /*Sleep during 2 seconds*/
}
```

Figure 6. Service method setVolume() for TVA

```
public void setVolume(unsigned int x) {
 IrControler con = new IrController();  /*Controller and */
 IrSignal sig = new IrSignal();    /*signal objects of Ir-API*/

 if (Power==OFF) ON(); /*Turn on when TV_A is off*/

 sig.setSignalType(VOLDOWN, TV_A); /*Volume down signal*/
 for (repeat_enough_times) {       /*Minimize the sound level*/
   con.sendSignal(sig);
 }

 sig.setSignalType(VOLUP, TV_A);   /*Volume up signal*/
 for (;x>0;x--) {                  /*Issue volume up x times*/
   con.sendSignal(sig);
 }
}
```

of volume-down, and then sends the signal of volume-up for x times. Moreover, the sound adjustment feature depends on the power feature, since it works only when TV_A is ON. Therefore, setVolume() contains an invocation of ON() in case the power is off. Thus, the service method becomes self-contained.

Supplementary Service: Getting Status

Some HNS applications may get the current status of an appliance, and then perform an appropriate action based on the status. A typical example is an energy-saving service, which stops a DVD player when a TV is turned off.

However, the communication among a user and a legacy appliance is basically *one-way* from the IrRC to the appliance. Hence, it is impossible for the external application to obtain the current status directly from the legacy appliance.

To cope with the problem, we implement, in the service layer, a supplementary feature that stores the *current state* of the appliance according to the history of service execution. Specifically, for every appliance, we prepare a database, called *state DB*, for storing the current values of primary properties the appliance. When a service method is executed, the values of the corresponding properties are updated in the state DB. We then deploy a service method getStatus()which returns the current state (i.e., the tuple of the property values) upon the request from the external applications.

Web Service Layer: Exporting Service Methods as Web-APIs

For every service method implemented in the service layer, we export the method to the HNS as a Web-API. In this article, we adopt the Web services (W3C, 2002), which is a standard SOA framework, for the service exportation. The interface definition of each service method is strictly typed by an XML-based language, called WSDL. An external application first interprets the interface definition, and then invokes a Web-API via network with appropriate parameters.

HNS Integrated Services

Once features of every legacy appliance are exported as Web-APIs, the legacy appliances can be used as *distributed components*. By

combining these components, the user can assemble various integrated services. Specifically, an integrated service can be implemented as a client application consisting of invocations of the Web-APIs and a control flow among the APIs. Note that the user can invoke Web-APIs with an arbitrary control flow, since every Web-API should be self-contained. For instance, let us consider the DVD theater service mentioned in Section INTRODUCTION. Figure 7 represents a sequence of Web-API invocations that can implement of the service.

The client application is installed on a remote PC or a handheld device owned by the user. It is also possible to install the integrated service on the home server. Then, the integrated service can be executed even from outside home. Also, the integrated service can be deployed as a new Web service to be reused for more sophisticated services.

IMPLEMENTATION: NAIST-HNS

Based on the proposed framework, we have implemented an actual HNS with the legacy appliances. The developed HNS is called NAIST-HNS.

Legacy Appliances Used

The following legacy appliances have been used in NAIST-HNS. Note that our system is composed of *multi-vendor* appliances:

Plasma display:
NEC PX-50XM2
DVD/HDD recorder:
Toshiba RF-XS46
Wireless LCD TV:
Sony KLV-17WS1
Ceiling light:
Panasonic HHFZ5310
Curtains with actuator:
NAVIO Powertrack
Air cleaner:
Hitachi EP-V12
Air circulator:
MORITA MCF-257NR
Power plug with IrRC:
HORIBA IS-100
Climate monitor (sensor):
IT Watchdogs WxGoos-1

Implementation of Legacy Adapter

Figure 8 depicts the overview of the developed legacy adapter for NAIST-HNS. In this implementation, we installed services for all appliances within a single PC. Note that this is just for convenience of the experiment. The services can be distributed among multiple PCs if necessary (see evaluation section for more details).

Technologies used for the system components are summarized as follows:

PC:
Celeron M360J, 512MB, 80GB, WinXP Pro
IrRC I/F:
Sugiyama Electron -- Crossam2+USB
IrRC Driver:
Serial COM library for Crossam2+USB
Ir-API:
Java Native Interface (JNI) Wrapper

Figure 7. Integrated service (DVD theater)

```
TV.setInput(DVD);            /*Set input mode for DVD*/
Light.setBrightness(1);      /*Minimize brightness*/
Curtain.Close();             /*Close curtain*/
Speaker.setInput(DVD);       /*Set input mode for DVD*/
Speaker.setChannel(5.1);     /*Set channel to 5.1ch*/
Speaker.setVolume(25);       /*Set volume level to 25*/
DVD.play();                  /*Play DVD*/
```

Service Layer:
J2SE 5.0
Web Service Layer:
Apache AXIS 1.3

The Crossam2+USB (Crossam, for short) adopted for IrRC I/F is a programmable infrared remote controller, which can be connected to a PC. Crossam can memory infrared signals of various legacy appliances, and can dispatch a signal in each button. The customization of Crossam is quite simple, and thus we can easily correspond to addition or replacement of legacy appliances.

Also, Crossam has a bundled serial port library written in C++. Using the library, the external program can send low-level commands, such as press a button, start and stop signal, to Crossam. Hence, we have used the bundle for IrRC Driver.

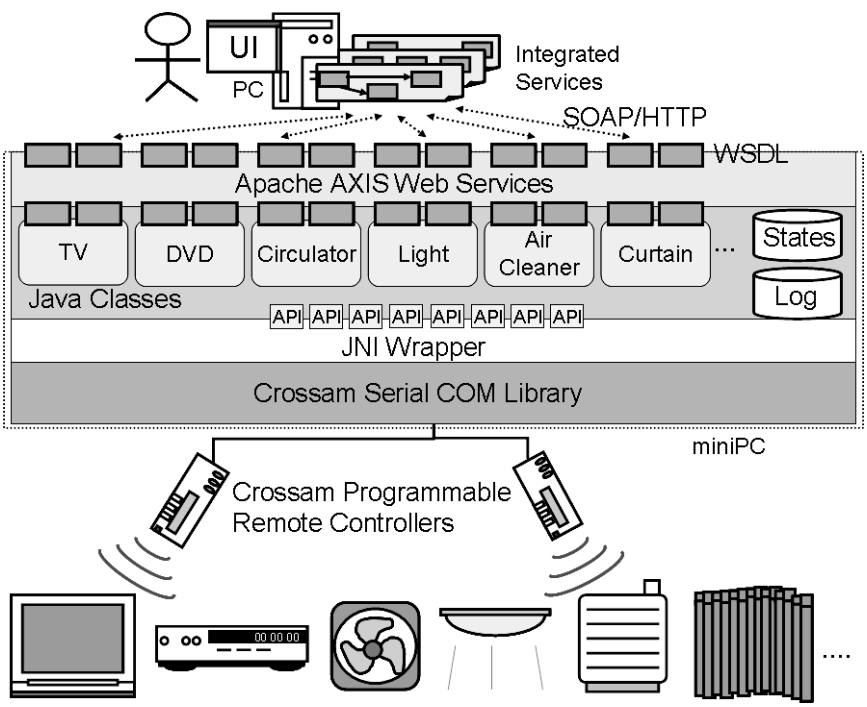
Next, we implemented Ir-APIs that can be invoked from Java programs by wrapping the Crossam library with Java Native Interface (JNI). The usage of JNI is due to consideration of the portability of Web services. The JNI wrapper comprised 130 lines of Java code.

The service layer was implemented in Java. For each appliance we constructed a Java class, where features of the appliance correspond to service methods. We constructed total 12 classes (four for the climate monitor and eight for other appliances) with 132 methods, comprising 1196 lines of code. Finally, the Java classes were deployed as Web services using Apache AXIS.

NAIST-HNS Integrated Services

We have implemented the following integrated services as client programs using Perl's SOAP::Lite module (Kulchenko & Reese, 2004).[2] Although the service scenarios seem to be sophisticated, they could be implemented in quite small programs, each of which comprises about 20-30 lines of code.

Figure 8. Overview of the HNS implementation



- **DVD Theater:** Integrating the TV, the DVD player, the speakers, the lights, and the curtains, the service automatically sets up the living room in a theater configuration. Upon a user's request, the TV is turned on with the DVD input, the curtains are closed, brightness of the lights are minimized, the speakers are configured for 5.1ch mode, and finally the DVD player plays the contents.
- **Air Cleaning:** Integrating the smoke sensor of the climate monitor, the air cleaner, the circulator, the service cleans dirty air in the room, automatically and efficiently.
- **Wakeup Support:** Integrating the speaker system, the lights, and the curtains, the service assists the user to wake up smoothly. Before 10 minutes of the given wakeup time, the speaker system and the lights are turned on with minimal sound volume and brightness. Then, the volume and the brightness are gradually increased, and set to the optimal at the wakeup time. Finally, the curtains are opened.
- **Auto illumination:** Integrating the climate monitor, the curtains, and the lights, the service always keeps the optimal lighting in the room regardless of time and weather.

Figure 9 shows a picture of our experimental room, where a user is trying to activate the DVD theater service using a user interface (discussed in the following section).

User Interfaces

We have also implemented various graphical user interfaces (GUIs) for human users to operate NAIST-HNS. Figure 10 shows three kinds of the developed interfaces.

Figure 10(a) shows a fancy interface written in Flash. This user interface displays a service menu on a TV in the room. Using a small remote controller, the user can choose an integrated service from the menu. Then, the user can activate and deactivate the service. By using this interface, it is no more necessary for the user to handle too many remote controllers for multiple appliances (see controllers on the table in Figure 9).

Figure 9. HNS experimental room

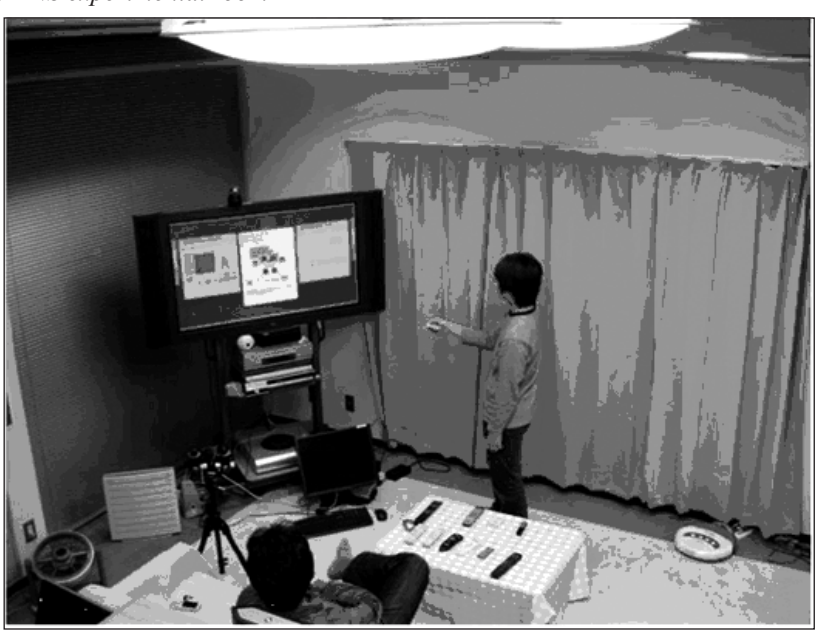


Figure 10(b) shows a GUI designated for mobile phones. This interface has been implemented as a Perl-CGI Web application. Using the mini browser in the mobile phone, the user can control appliances and services even outside home. Thanks to the service and Web service layers of the proposed architecture, it was possible to implement the Web Application quite efficiently based on a typical MVC (model-view-controller) approach. That is, Web services, HTML templates, and the perl-CGI scripts correspond to the model, view and controller, respectively. Specifically, every CGI script was implemented so as to perform the following common steps:

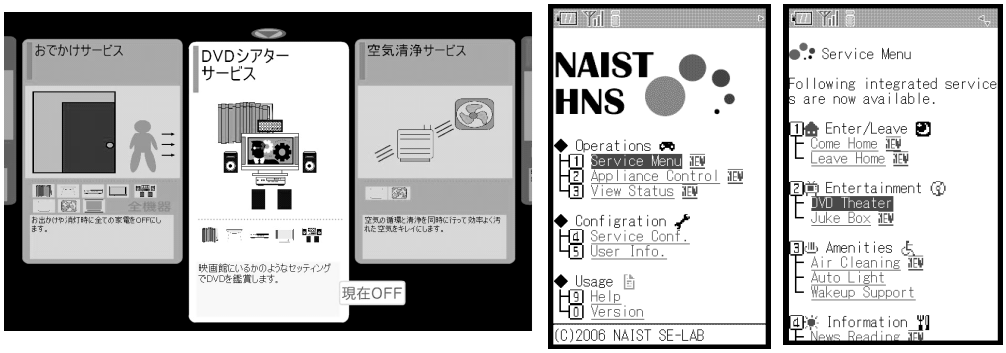
Step 1: On receiving an HTTP request, parse the request and obtain the user inputs.
Step 2: Based on the inputs, activate Web services of appropriate legacy appliances.
Step 3: Get the current state of the appliances through getStatus() Web-API.
Step 4: Generate a status message from the current state and responses of the Web services.
Step 5: Embed the message in the prepared HTML template, and output the HTML text as a HTTP Response.

Finally, Figure 10(c) shows light-weight GUIs developed as standard widget applications for the PC. The GUIs were written in Perl/Tk, operating on both Windows and Linux PCs. For the implementation, we basically follow the above common steps (Steps 2, 3, and 4). As a result, the size of application became quite small. For instance, the number of statements of CurtainRemocon.pl (left of Figure 10(c)) was as small as 28.

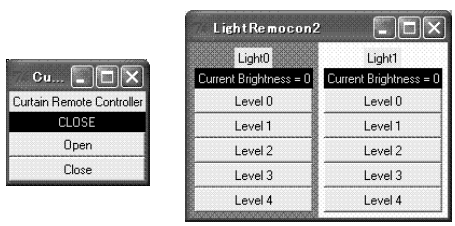
EVALUATION

In this section, we evaluate the proposed framework from several viewpoints.

Figure 10. User Interfaces for NAIST HNS



(a) Flash Interface (b) Mobile Phone Interface



(c) PC Application Interface

Achievements of Requirements

We first evaluate if the proposed framework satisfies requirements R1, R2 and R3 (see requirements section).

The integrated services can be implemented just by combining Web-APIs deployed with Web services. Since the service layer was designed to be vendor-neutral and self-contained (see conditions S1 and S2), the developer can invoke the Web-APIs in any combinations and any order, without concerning the underlying IR signals or vendor-specific issues. Also, as the lines of code of the developed integrated services indicate, the developer can create their own integrated services without much effort. Thus, R1 is achieved.

Requirement R2 is surely satisfied as we demonstrated it with NAIST-HNS. We have built up the whole NAIST-HNS by generic PCs and commercial universal remote controllers.

The proposed framework applies to a variety of legacy appliances, as long as the appliance has an infrared remote controller (IrRC), and its signals are compatible to the proposed legacy adapter. Thus, requirement R3 is satisfied. As for appliances that do not have the IrRC, we can use a power relay device with IrRC (HORIBA, 2000) for providing simple services like power on and off.

Performance

One of major concerns in any SOA-based system is *performance*. We have measured *response time* of each integrated service developed in Section IMPLEMENTATION, in order to clarify the overhead posed by the proposed framework. There are two kinds of response time: *device response time (DRT)* and *service response time (SRT)*. The DRT is the time physically taken for the legacy appliances to execute features required in the service. For instance, turning on our DVD/HDD recorder required DRT about 30 seconds for initializing the system. On the other hand, the SRT is the time purely devoted for processing the service layer with Web services, which is interesting for us here.

Table 1 shows the result. Each row represents the corresponding response time in milliseconds. [3] The last row represents the number of Web-APIs executed in each integrated service.

It can be seen in the table that the total response time varies from service to service. This is because the different services use different sets of legacy appliances. However, SRT has a strong correlation with the number of Web-APIs executed, regardless of the features executed by the Web-APIs. This is because the time taken for the middleware to process Web service messages became a dominant factor for SRT. SRT for each Web-API invocation varied from 0.22 seconds to 0.65 seconds. Hence, it can be said that the proposed framework suffers from quite a little overhead. Hence, we should count the overhead carefully when developing services that require time-sensitive or real-time transactions.

However, we are optimistic for the overhead in the service layer. The overhead is basically due to performance of the current WSDL implementations in marshalling XML data to and from messages. This problem will likely be alleviated by future WSDL implementations as the technology matures.

Table 1. Response time of integrated services

	DVD Theater	Air Cleaning	Wakeup	Illumination
SRT (msec)	3,563	2,613	4,905	676
DRT (msec)	39,350	3,000	19,600	1,250
Total (msec)	42,913	5,613	24,505	1,926
# of Web-APIs executed	6	4	16	3

Productivity

In the proposed architecture, vendor (or device)-specific operations of legacy appliances are completely encapsulated within the service layer, whereas vendor-neutral features can be accessed as Web services. Currently, many programming languages support SOAP and WSDL libraries invoking Web services. Therefore, what required for the application/service developer is just to know the location of the Web service definition (i.e., WSDL), and the names and parameters of desired Web-APIs.

After that, for each invocation of an appliance feature, the developer writes code just for two operations: (1) instantiate Web service, and (2) invoke the service object. In case of Perl, these operations can be written just within two lines. As for Java, the IDE (integrated development environment), such as *Eclipse* or *NetBeans*, typically has a feature that automatically generates related skeleton code for Web service invocations. The developer just adds a few lines to the skeleton code. Thus, our idea of exploiting Web services significantly contributes to the high productivity of new services and applications using the legacy appliances.

Using the Web service integration framework such as BPEL4WS (Weerawarana & Curbera, 2002) may enable more efficient service creation. The further discussion of the applicability of BPEL4WS to HNS applications/services is left to our future work.

Extendibility

The proposed architecture is not strictly bound with the legacy appliances only. The same architecture can also be applied to the emerging networked appliances. This is done by replacing the IR device layer in Figure 4 with a certain HNS protocol stack and control APIs of the networked appliances. The point is to deploy the service and Web service layers on top of the proprietary device implementation. Since different appliances can be *uniformly* managed as Web services, it is easy to add new appliances and new services.

Another merit in extendibility is that one can integrate legacy appliances with the external Web services in the Internet. That is, it is no more necessary to distinguish the home appliances from the conventional information services and resources in the Internet. This is a great benefit in creating more intelligent and sophisticated HNS services. The followings are interesting examples which integrate legacy appliances using news and stock Web services in the Internet:

News flash: As soon as important news arrives, turn on the TV with the selected channel.
Stock alarm: When a stock price rises, notify it of users by ringing chimes in the house.

Development of more sophisticated services is left to future work.

Portability

Since all features of legacy appliances are wrapped within Web services, applications using the legacy appliances can be ported to various platforms and languages. As seen in the previous section, the service layer of all appliances in NAIST-HNS was implemented in Java on the Windows platform. However, this fact does not require client applications or integrated services to use specific language and platform (e.g., Java and Windows). Indeed, we have implemented client applications and integrated services in Windows, Linux, and Sun Solaris platforms. They worked as expected. We have also tested various languages for implementing the integrated services. The languages tested include Perl, PHP, Java, C# .Net, VB, Delphi (Pascal). Thus, we were able to achieve the programmatic interoperability taking the full advantage of Web services. This characteristic is quite unique compared with other standards for home network systems and networked appliances.

Maintainability

The service layer of the proposed framework plays an important role, which achieves a *loose-coupling* between the HNS applications (as service consumers) and the appliances (as service providers). As long as the interface

definition of the service is not changed, one can change, replace or update the appliances without influencing the service consumers. Therefore, the HNS based on the proposed framework is quite resilient for the system modification and evolution.

Limitations

A technical limitation of the proposed framework is in the reliability of the physical channel between IrRC/IF and the appliance. As discussed in the supplementary service section, the communication from the PC to a legacy appliance is basically one-way. Hence, it is not easy to confirm whether or not an infrared signal is successfully received by the appliance. If the signal is lost, inconsistency between the state DB and the actual status of the appliance occurs, which may lead the integrated service to malfunction. Therefore, we need to guarantee the reliability of the physical channel at all cost.

Fortunately in our framework, the service layer can be distributed within multiple PCs. Hence, in an extreme case, we can assign a PC for *every* appliance so that the IrRC is close enough to the appliance. A smarter approach is to assign a PC for each group of *neighbor* appliances. The user has to choose a reasonable layout considering reliability requirement, cost for PCs, a floor plan and objects in the room.

User authentication and security management are beyond the scope of this article, but are important issues in practical usage of the HNS. We conduct these problems in our future work.

RELATED WORK

Loke (Loke, 2003) firstly modeled networked appliances as Web services, and proposed a workflow engine, called *Decoflow*, which prescribes integrated services using BPEL4WS. However, the method regards each networked appliance just as a *black box* with an open interface. Also, it does not mention the legacy appliances. The proposed framework can complement the method, which would allow efficient creation and management of the integrated services.

SOA with Web services is expected as a powerful means to achieve *legacy migration*. For instance, Lewis, Morris, O'Brien et al. (2005) presented a framework called SMART, which assists organizations in analyzing legacy capabilities for use as services in SOA. Zhang and Yang (2004) proposed a method based on a code analysis, which facilitates legacy code extraction for Web service construction. However, most existing techniques including the above are addressing the legacy migration of *enterprise* systems. We believe that our original contribution was to show a concrete framework for legacy migration in the new domain, that is home network system, exploiting the essence of SOA with Web services.

Another interesting issue is the *feature interaction problem* (Kolberg et al., 2003; Metzger, 2004; Weiss & Esfandiari, 2006), which is the functional conflicts among multiple HNS services on the appliances or environmental factors. Nakamura, Igaki and Matsumoto (2005) formalized the feature interaction problem and presented a conflict detection method in the context of the HNS. We are currently implementing the method on the developed system.

CONCLUSION

This article presented a framework that adapts the legacy home appliances to the emerging home network system. The proposed framework extensively adopted the concept of SOA. Features of the legacy appliances are exposed as self-contained Web-APIs with Web services. We also implemented the actual HNS with multi-vendor legacy appliances, as well as several integrated services.

Our future work includes the security issues and the feature interaction management. We also plan to investigate a method that supports *non-expert users* in creating integrated services easily.

ACKNOWLEGMENT

This research was partially supported by: the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Young Scientists (B)

(No.18700062), and Grant-in-Aid for 21st century COE Research (NAIST-IS —Ubiquitous Networked Media Computing).

REFERENCES

DLNA (2006). *Digital Living Network Alliance* http://www.dlna.org

ECHONET (2006). ECHONET Consortium http://www.echonet.gr.jp/

Geer, D. (2006). Nanotechnology: The growing impact of shrinking computers. *IEEE Pervasive Computing*, *5*(1), 7-11.

Hitachi Home & Life Solutions Inc. (2003). HORASO Network Service. http://ns.horaso.com/index.html

HORIBA (2000). Tsuichau-mon – Light Right. http://www.jp.horiba.com/products_e/hip08/hip08_04.htm#6

Igaki, H., Nakamura, M., & Matsumoto, K. (2005). A Service-Oriented Framework for Networked Appliances to Achieve Appliance Interoperability and Evolution in Home Network System (short paper). In *Proceedings of the International Workshop on Principles of Software Evolution (IWPSE 2005)*, 61-64.

Kaden Control Lab. (2006). "AVT Series", http://d-purasu.hp.infoseek.co.jp/

Kolberg, M., Magill, E. H., & Wilson, M. (2003). Compatibility issues between services supporting networked appliances. *IEEE Communications Magazine*, *41*(11), 136-147.

Kulchenko, P., Reese. B. (2004). SOAP::Lite - Client and server side SOAP implementation http://www.soaplite.com/

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

Loke, S. W. (2003). Service-Oriented Device Echology Workflows. In *Proceedings of the1st Int'l Conf. on Service-Oriented Computing (ICSOC2003)*, LNCS2910, 559-574

Matsushita Electric Industrial Co., Ltd. (2005). "Kurashi Net", http://national.jp/appliance/product/kurashi-net/

Metzger, A. (2004). Feature interactions in embedded control systems. *Computer Networks*, *45*(5), 625-644.

Nakamura, M., Igaki, H., & Matsumoto, K. (2005). Feature Interactions in Integrated Services of Networked Home Appliances -An Object-Oriented Approach. In *Proceedings of the Int'l. Conf. on Feature Interactions in Telecommunication Networks and Distributed Systems (ICFI'05)*, (pp. 236-251).

Nakamura, M., Tanaka, A., Igaki, H., Tamada, H., & Matsumoto, K. (2006). Adapting Legacy Home Appliances to Home Network Systems Using Web Services. In *Proceedings of the of IEEE International Conference on Web Services (ICWS2006)*, (pp. 849-858).

NANO Media Inc. (2005). "App-rimo-con", http://www.nanomedia.jp/english/service/s02.html

Papazoglou, M. P., & Georgakopoulos, D. (2003). Service-oriented computing. *Communications of the ACM*, 46(10), 25-28.

Smith, D. J., & Meyers, B. C. (2005). Exploiting Programmatic Interoperability: Army Future Force Workshop. *Technical Note CMU/SEI-2005-TN-042*, Software Engineering Institute.

Toshiba (2005). "Toshiba home network – Feminity", http://www3.toshiba.co.jp/feminity/feminity_eng/

W3C (2002). "Web Service Activity", http://www.w3.org/2002/ws/

Weerawarana, S., Curbera, F. (2002). "Business process with BPEL4WS: Understanding BPEL4WS, Part1", http://www-106.ibm.com/developerworks/webservices/\\library/ws-bpelcol1/

Weiss, M., Esfandiari, B. (2005). On feature interactions among web services. *International Journal of Web Services Research*, *2*(4), 22-47.

Zhang, Z., & Yang, H. (2004). Incubating services in legacy systems for architectural migration. In *Proceedings of the11th Asia-Pacific Software Engineering Conference (APSEC'04)*, (pp. 196-203).

ENDNOTES

[1] The IrDA, which is often used for exchanging data among PCs and handheld devices, is completely different from the IrRC, although both protocols use the infrared for the physical layer.

[2] We did not use BPEL4WS for implementing the integrated services, since BPEL4WS was somehow exaggerated for our current HNS implementation. Further investigation of its applicability is left for our future work.

[3] The time taken for each Perl client to prepare SOAP stubs from the corresponding WSDLs had been excluded from the data.

Masahide Nakamura received the BE, ME, and PhD degrees in information and computer sciences from Osaka University, Japan, in 1994, 1996, 1999, respectively. From 1999 to 2000, he has been a post-doctoral fellow in SITE at University of Ottawa, Canada. He joined Cybermedia Center at Osaka University from 2000 to 2002. From 2002 to 2007, he worked for the Graduate School of Information Science at Nara Institute of Science and Technology, Japan. He is currently an associate professor in the Graduate School of Engineering at Kobe University. His research interests include the service-oriented architecture, Web services, the feature interaction problem, V&V techniques and software security. He is a member of the IEEE, ACM and IEICE.

Akihiro Tanaka received the BE in information and mathematical from Nara University of Education, Japan in 2005. He received ME degree from Nara Institute of Science and Technology, Japan in 2007. He is currently working for Hitachi, Ltd. His research interests include service oriented architecture, Web service, home network system, legacy appliances.

Hiroshi Igaki received the BE degree (2000) in Department of Electrical and Electronics Engineering from Kobe University, Japan, and the ME degree (2002) and DE degree (2005) in information science from Nara Institute of Science and Technology, Japan. He is currently an assistant professor of Graduate School of Engineering at Kobe University. His research interests include communication support in software development, web services and service-oriented architecture. He is a member of the IEEE and a member of the ACM.

Haruaki Tamada received the BE and ME in information and communication engineering from Kyoto Sangyo University, Japan in 1999, 2001. He received DE degree from information science from Nara Institute of Science and Technology, Japan in 2006. He is currently an assistant professor in Graduate School of Information Science, Nara Institute of Science and Technology, Japan. His research interests include software security, software measurement. He is a member of the IEICE, IPSJ and IEEE.

Ken-ichi Matsumoto received the BE, ME, and PhD degrees in Information and Computer sciences from Osaka University, Japan, in 1985, 1987, 1990, respectively. Dr. Matsumoto is currently a professor in the Graduate School of Information Science at Nara Institute of Science and Technology, Japan. His research interests include software metrics and measurement framework. He is a senior member of the IEEE, and a member of the ACM, IEICE, IPSJ and JSSST.