Evaluation of SONICA Compared with UPnP and Jini

Hiroshi Hayakawa[†], Takahiro Koita[‡]and Kenya Sato[‡]

[†]Graduate School of Engineering, Doshisha University, dtf0712@mail4.doshisha.ac.jp
[‡]Department of Information Systems Design, Doshisha University, {tkoita,ksato}@mail.doshisha.ac.jp
1-3 Tatara Miyakodani, Kyotanabe-City, Kyoto, 610-0321, Japan

ABSTRACT

The current move towards a ubiquitous network society is encouraging companies to develop various items in preparation for networking. Accordingly, home appliances that operate by themselves are beginning to be equipped with communication interfaces, and these appliances are leading the way and illustrating next generation services. Thus, some platforms for constructing home networks have been proposed, such as DLNA, Jini, and HAVi. Although these platforms are expected to play a large role in the spread of home networks, most of them are based on PC-centric networks. Considering the limited resources of embedded systems, customizability for the end-user, and other specific home appliance requirements, this situation raises issues of software and hardware complexity, cost, power consumption, and so on. The complexity and latitude of services provided largely depends on the architecture of device interaction. In this paper, we propose a new platform named SONICA (Service Oriented Network Interoperability for Component Adaptation). SONICA adopts service oriented interoperability using HTTP methods and a more simple and flexible architecture for device interaction. We also show the usefulness of SONICA as a platform for home networks by conducting evaluations with the existing platforms; UPnP and Jini.

Keywords: Home Network, UPnP, Jini, embedded systems, interoperability, ubiquitous

1 INTRODUCTION

The current move towards a ubiquitous network society is encouraging companies to develop various items in preparation for networking. Accordingly, home appliances that operate by themselves are beginning to be equipped with a communication interface, and these applicances are leading the way and illustrating next generation services. Thus, some platforms for constructing home networks have been proposed, including DLNA (Digital Living Network Alliance)[1], Jini[2], [3], and HAVi (Home Audio/Video Interoperability)[4]. DLNA aligns industry leaders including Microsoft, Intel, Panasonic, and SONY, and is becoming the de facto standard. DLNA adopts UPnP (Universal Plug & Play) Device Architecture 1.0[5] for device interoperability. Jini, which was proposed by Sun Microsystems, is designed for use in dynamic operating environments like pervasive networks where the state of devices changes frequently. Although these platforms are expected to play a large role in spread of home network, most of them are based on PC-centric networks. Considering that the available resources of each home appliance are limited and heterogeneous, the resources required for device interaction should be as small as possible in order to support many devices. Moreover, the complex manner of the interactions may raise issues such as cost and power consumption. Ubiquitous computing is a paradigm shift where technology becomes virtually invisible in our lives, and the goal is that people can access services without any conscious effort. However, some end users want to customize or build up services as they please. From a barrier-free society view, we should focus on customizability for the end-user. The complexity and latitude of provided services is tied to the architecture of device interaction. Consequently, the architecture of device interaction should be kept simple and flexible. In this paper, we propose a new platform named SONICA (Service Oriented Network Interoperability for Component Adaptation). SONICA adopts service oriented interoperability using HTTP methods and a more simple and flexible architecture for device interaction. The next section discusses the requirements of architecture for device interaction on a home network. In the 3rd section, we talk about UPnP and Jini as major platforms. In the 4th section, we discuss the design of SONICA as a solution for issues found in existing platforms. We evaluate SONICA against existing platforms in the 5th section. In the 6th section, we show a practical system constructed on SONICA, and then conclude with a summary in the 7th section.

2 REQUIREMENTS

In this section, we outline the requirements of architecture for device interaction.

2.1 Simplicity

Most devices connected to home network are embedded systems, which do not have so much resources as a PC has. Consequently, the resources required for device interaction should be as simple as possible in order to support many devices. Moreover, complexity in device interaction may cause issues such as cost, power consumption, and an increase in network traffic.

2.2 Compatibility

Some devices connected to the home network have a long service life, whereas some will be replaced by new ones in a short period. Even if some of the devices are replaced, the interactions between the new devices and the other appliances must be guaranteed. Generally, products produced by multiple vendors are used together and must work in concert with each other. To maintain interaction compatibility, updating the software of all devices may be required. However, as the number of devices connected to the network increases, it becomes increasingly difficult for end-users to maintain devices and update their software. Increased network traffic is also a concern. Compatibilities are largely determined by interfaces of interaction. Consequently, the interaction interfaces should have high latitude and flexibility to ensure long availability and vendor-independence.

2.3 Customizability

Interaction among devices facilitates various kinds of services. However, some end users want to customize or build up services to suit their own needs. For example, a user may want to recieve or be informed of messages by voice or sound, while others may desire subtitles on their TV. Customizability that allows the end-user to adjust services as they like is neccessary for barrier-free environments. There are also possibilities to generate a variety of other services. The customizability of services depends on the complexity and latitude of interaction architecture. Consequently, the architecture for device interaction should be kept simple and flexible.

3 EXISTING PLATFORMS

Some platforms have already been proposed for constructing home networks. We introduce UPnP and Jini in this section.

3.1 UPnP (Universal Plug & Play)

UPnP was proposed by Microsoft who aimed to apply the concept of Plug & Play to network systems. UPnP is adopted as a device control technology for DLNA led by Microsoft, Intel, SONY, Panasonic, and other industrial leading vendors. The technologies underlying UPnP Device Architecture include SOAP[6]–[9] as a remote procedure call (RPC) mechanism. SOAP allows devices to offer complex and advanced services. However, each UPnP device must have a XML parser and must parse all communication messages described in XML. This results in inefficiency and requires additional software resources. In addition, because of the static interfaces of SOAP, it is necessary to standardize a method to control a new device upcoming in the future, whose function is currently undefined, and to update already installed software modules in the currently undefined device.

3.2 Jini

Jini was proposed by Sun Microsystems in 1999; they aimed at a network plug & play especially for dynamic networks where the resources are more volatile. For the last few years, Jini has been attracting attention once again because of its new license system and improvements in the performance of embedded systems. In the Jini network, a server registers instances of its service interface and codebase annotation using a lookup service. Using a template that consists of service's interface and attributes, a client retrieves services through the lookup service. The actual implementation of a ServiceItem, such as an instance of service or may proxy, will be downloaded from the location specified in the codebase annotation on the ServiceItem that the server supplied to the lookup service. Thus, the service's instance must be available on the network, and the codebase annotation has to be set to point to its location. The Jini network system can automatically set up remote services. However, in order to lookup services and re-serialize a marshalled object, clients have to know the service's interface in advance. This means that the Jini network system requires interface standardization or to install a new interface when needed. To transfer objects via the network, management systems such as a lookup service and a codebase are needed; this require many resources. In addition, the increase in network traffic is also a concern. Although most of Jini software services are written in Java and their communications are based on the Java RMI (Remote Method Invocation) mechanism, Jini specifications are not constrained to Java.

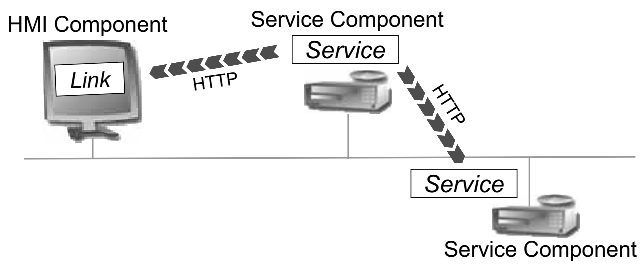

Figure 1: Device Interaction

4 SONICA

To solve the above issues, we have proposed and developed SONICA (Service Oriented Network Interoperability for Component Adaptation) for the control and management of home appliances over ubiquitous networks. SONICA is based on simple HTTP protocols with WebDAV technology[10], which is a multimedia application platform for use in embedded system. Interaction is based on the services of each device.

4.1 Device Interaction

In SONICA, devices interact with each other using the partner's service. Figure 1 shows an outline of device interaction in SONICA. When a device (service component) is connected to the network, device information (including link information for performing services) offered by the connected device is sent to other devices. In this case, only device information is sent and the service module is never sent. Another device (the HMI component) generates a menu with the link information received from the connected service component.

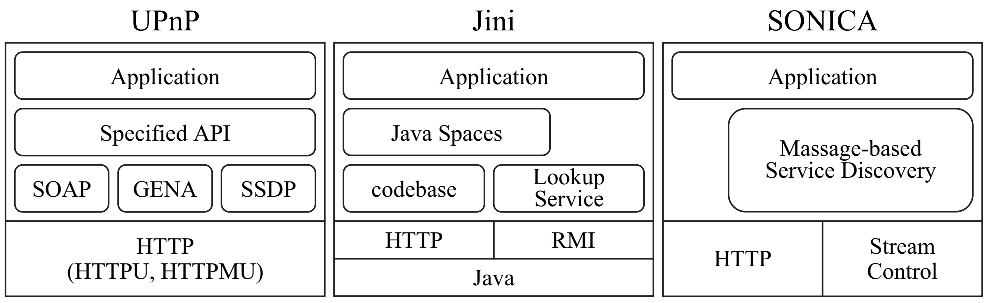

Figure 2: Module Structure of Each Platform

When a user selects a service on the menu, the HMI component that has the menu sends a message to the requested service component, and the service component provides the requested service. The user does not have to know where the service is provided from, only that it is available on the HMI component. Devices such as HMI components do not need to implement complex software modules such as the service mechanism, running processes, or other machine architectures. With this kind of information based on the SONICA mechanism, there is no need to define service control messages in advance, and there is no difference for any device, whether the method of accessing a device is defined or undefined. As a result, the system can be easily constructed with Plug & Play while effectively using the hardware resources, and network traffic can be suppressed. UPnP and HAVi adjust to undefined devices by loading or updating software modules, however, it is difficult to support many kinds of devices because the device's resources are limited.

The SONICA interaction mechanism is based on HTTP verbs (e.g. GET, POST, PUT, and DELETE). Services based on HTTP verbs are simple and can be easily used with other devices without specific software modules. In addition, we use WebDAV technology to achieve interactions between devices.

4.2 Components

In SONICA, the system is composed of a human machine interface component (HMIC) as a GUI (graphical user interface) and other components called service components (SC). An SC consists of an HTTP server function and services provided by equipment like DVD recorders, tuners, audio amplifiers, and video cameras. Those services are provided to a user after the user makes a request to an HMIC. A single physical device may contain one or more components. Each component keeps tabs on the device components when a device is connected or removed. To use limited bandwidth efficiently, a bandwidth management mechanism is needed. A resource manager reserves the required bandwidth and assigns the bandwidth according to the device's demands. The SC also becomes a resource manager or a DHCP server in the network.

An HMIC that includes an XML browser function works as

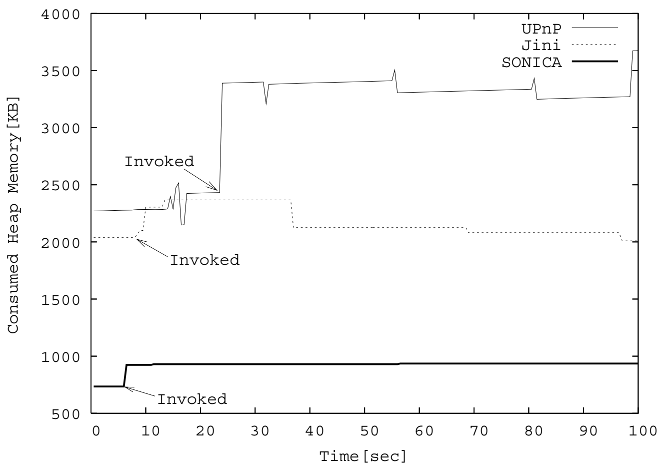

Figure 3: Heap Memory Consumption

an interface. A user can operate SCs connected to the network using the interface. The HMIC generates a menu from the device information offered by the SC. It is easy to control the components with user interfaces that have a high user affinity, such as GUI. However, it is possible to accept a simple user interface without graphical images (e.g. an ordinal remote controller) to control the SCs.

5 ARCHITECTURE EVALUATION

In this section, we evaluate device control architecture of UPnP, Jini and SONICA. Figure 2 shows the module structure of each platform.

5.1 Implementation of UPnP, Jini and SONICA

We implemented simple services for the evaluation of each platform; UPnP, Jini, and SONICA. Each implemented service just returns the current time using java.util.Date class. For UPnP, we used CyberLink[11] for Java as protocol stacks. In this evaluation, we adopted Xerces as an XML parser for CyberLink, although KXML is also available.

We used Java RMI as a communication mechanism for Jini service. To exclude the resources for service discovery, we

Table 1: Evaluation Results of Each Platform

Platform	UPnP	Jini	SONICA
Maximum Consumed Heap Memory	more than 3000 KB	2367.5 KB	935.8 KB
Response Time (Average)	1693.9 ms	16.9 ms	10.2 ms

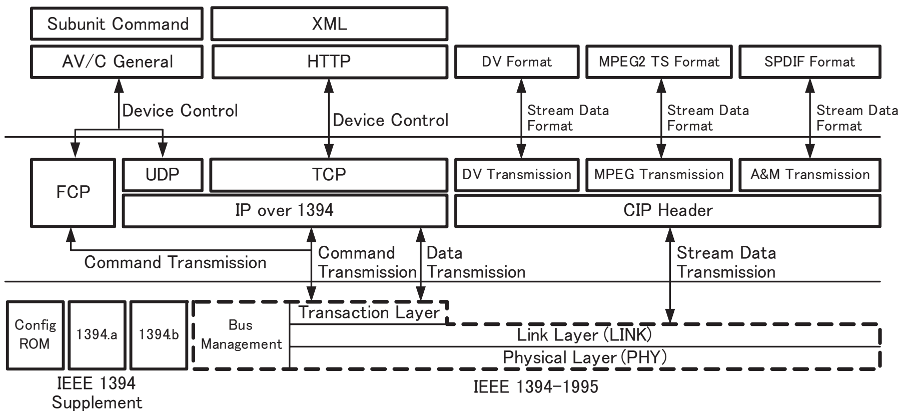

Figure 4: SONICA Protocol Stack over IEEE 1394

did not use a lookup service but rather the rmiregistry we describe later. We also used a local file system for codebase and not a HTTP server.

We designed and implemented a tiny HTTP server for SONICA SC. The server is based on a thread pooling model, and can handle HTTP GET and POST requests as device control triggers. To be fair, the adopted platforms and implemented services are written in pure Java. Evaluation is performed on Java 5.0.

5.2 System Evaluation

We measure the consumed heap memory size and response time for UPnP, Jini, and SONICA. Figure 3 shows variation in heap memory usage when each service is invoked. Table 1 shows the maximum size of the consumed heap memory and the response time between sending the request and receiving a response. Usage of heap memory is measured every 500 ms for 100 seconds using jstat, which is a Java VM statistic data monitoring tool. Measurements are taken in two situations: when the service is waiting and running. The values shown as the usage of heap memory are sums that are delivered by multiplying the consuming rate by the reserved size in the following three generations: NEW Generation (includes the Eden Range and Survivor Range); OLD Generation; and Permanent Generation. The value shown as Jini includes not only resources for the implemented service itself but also for the rmiregistry. The service discovery mechanism is implemented in quite different ways for each platform, for example SSDP (Simple Service Discovery Protocol) on UPnP and the Lookup Service on Jini. To make a fair comparison, the resources used for service discovery are not included in this evaluation.

Compared with the other two platforms, UPnP has a 1693.9 ms response time. This is attributed to the cost of parsing each communication message described in XML. UPnP also consumes more than 3000KB of heap memory for the XML parser. Jini works well in the area response time, but on the other hand, it needs much more memory for the rmiregistry. In the RMI communication sequences, marshalled objects are transferred so the network traffic is thought to be heavier. On the first request, marshalled objects are transferred to the client. However, on consecutive accesses, the cashed service works in the client VM. So the Jini response time changes whether it is the first request or not. In this evaluation, we used rmiregistry and a local file system for the codebase as Jini implementation. However, on a real Jini system with a lookup service and a http codebase, many more resources are needed, and the response time is thought to be longer. Compared with UPnP and Jini, the service with SONICA responds rapidly, and the consumed memory size is less than 1MB. These results lead us to the conclusion that SONICA is more suitable for embedded systems with limited resources.

6 EVALUATION THROUGH APPLICATION

To evaluate SONICA in a real application environment, we implemented a video camera controlling system with SONICA on some PCs (Fedora Core 3 with customized kernel based on 2.6.9, Pentium 4 3GHz).

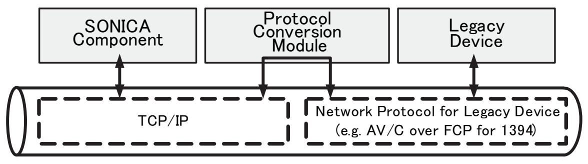
SONICA Component
Protocol Conversion Module
Legacy Device
TCP/IP
Network Protocol for Legacy Device (e.g. AV/C over FCP for 1394)

Figure 5: Legacy Device Support Mechanism

6.1 SONICA over IEEE 1394

In this study, we implemented SONICA over IEEE 1394 (a.k.a. iLink or FireWire)[12]–[14] as a network interface, because the Ethernet is not capable of supporting the transport of real-time multimedia streaming data over the network, which is an essential function for home AV appliances. The IEEE 1394 is a serial interface that supports up to 800Mbps data transmission, and has an isochronous transmission mode for real-time streaming data, as well as an asynchronous transmission mode. The IEEE 1394 allows SONICA to construct AV appliances network without a PC.

Figure 4 shows the protocol stack for SONICA. SONICA constructs an IP network using the IPover1394[15] protocol to achieve high affinity with a PC and the Internet. To achieve QoS guarantees in the network, the isochronous transmission mode is processed in the low layers up to link layer and can be accelerated by hardware.

6.2 Supporting Legacy Device

A legacy device that does not correspond to SONICA can made be available using a protocol conversion module (PCM). Although PCM works as proxy, it appears SC form other SONICA corresponding devices. Figure 5 shows how a legacy device that uses AV/C commands is supported. In SONICA, HTTP is adopted to transmit requests and responses. PCM converts HTTP requests and responses into an original command acceptable for a legacy device. HTTP messages can coexist with other control protocols like FCP (function control protocol) or data transmission because IPover1394 uses an asynchronous transmission mode. Thus, if a device using a PCM for AV/C commands is connected to the network, all devices connected to the network can use the device as a PCM function.

6.3 Camera Application with SONICA

Figure 6 shows an overview of the communication procedure within SONICA network. The system consists of a HMIC, an SC as a content manager, and an IIDC PCM as PCM to control the IEEE 1394 video camera (Apple iSight).

A HMIC generates a menu from device information offered by the SCs. For simplicity, the HMIC directly access the SC's control menu. The content manager offers a service to save images sent from other components. IIDC PCM works as a proxy that converts HTTP requests and responses to IIDC requests and responses for IEEE 1394 digital cameras. The IIDC PCM allows iSight to connect to the SONICA network. We used iSight as a legacy device. In order

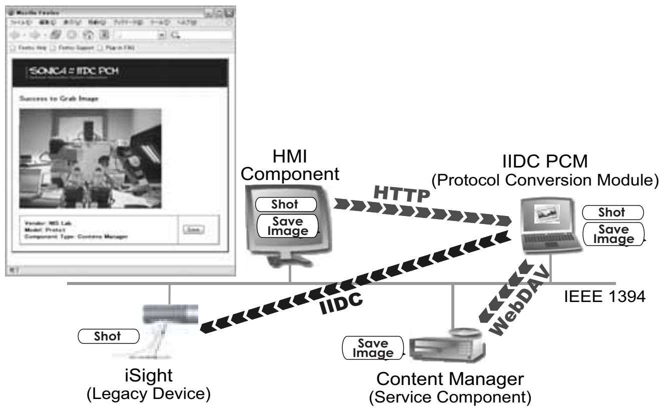

Figure 6: Communication Procedure

to use the IEEE 1394 digital camera, we developed a library named libdc1394j that controls the library called libdc1394 from Java.

On Linux, libdc1394 is available as an IIDC control library. Libdc1394 is described in C Language and is impossible to use directly in Java. To use the library directly in Java, we implemented a shared library that includes a controlling mechanism that is described in the native language, and a stab that loads this library from Java using JNI (Java native interface). This stab can be used as if it is the Java library. Figure 7 shows a libdc1394j software stack. Libdc1394j contains a stab which is an interface of libdc1394 and the original libraries. The original libraries include the image format translation method implemented in the native language. In libdc1394, error handling is strongly dependent on applications, however, it is difficult to manage errors that occur in the native methods. Thus, libdc1394j manages errors in the native methods, and then informs the Java applications. Call by reference is heavily used in the original source code and JNI does not define the passing of pointer variables. In the Java application, we handle the pointer variables as integer variables.

Below is the working sequence of this system. (1) A user accesses the IIDC PCM control menu that shows the functions and legacy components available now. (2) The user selects iSight from the menu. Using libdc1394j, the IIDC PCM allows iSight to take a picture. (3) The captured picture and the components providing the 'save' function are shown in the HMIC browser. (4)When the user selects the content manager to save the image, IIDC PCM uses the content manager's 'save function' and transfers the picture using WebDAV. (5) The user accesses the content manager's control menu. (6) The user confirms that the transferred image has been saved.

7 CONCLUSION

In this paper, we have proposed a new platform named SONICA, which can solve the performance and memory consumption issues on existing platforms (e.g. UPnP and Jini). SONICA designed for service oriented interoperability is based

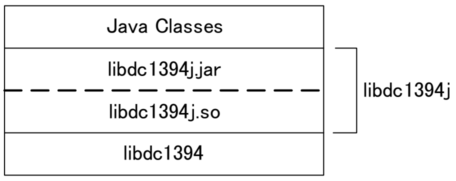

Figure 7: Libdc1394j Software Stack

on HTTP related protocols and WebDAV technology. Therefore, the architecture is simpler and more flexible than those of the existing platforms. We measured response time and memory consumption for UPnP, Jini and SONICA, and SONICA responded more quickly and consumed less memory than the other technologies. To demonstrate the advantages of SONICA, we implemented and evaluated the SONICA system over IEEE 1394 network to control a legacy video camera through the protocol conversion module. Our evaluation showed that SONICA is suitable for embedded systems even with legacy devices because of its simple and flexible architecture.

REFERENCES

[1] DLNA (Digital Living Network Alliance), Overview and Vision White Paper 2006 (2006). http://www.dlna.org/about/dlna_white_paper_2006.pdf

[2] Sun Microsystems, Jini Network Technology. http://www.sun.com/software/jini/.

[3] Jini.org, Jini Standards. http://www.jini.org/standards/.

[4] Home Audio Video Interoperability (HAVi) Organization, White Paper, HAVi, the A/V digital network revolution. http://www.havi.org/pdf/white.pdf.

[5] UPnP Forum, UPnP Device Architecture Version 1.0 (2000). http://www.upnp.org/download/UPnPDA10_20000613.htm.

[6] W3C Recommendation, SOAP Version 1.2 Part 0: Primer (2003). http://www.w3.org/TR/soap12-part0/.

[7] W3C Recommendation, SOAP Version 1.2 Part 1: Messaging Framework (2003). http://www.w3.org/TR/soap12-part1/.

[8] W3C Recommendation, SOAP Version 1.2 Part 2: Adjuncts (2003). http://www.w3.org/TR/soap12-part2/.

[9] W3C Recommendation, SOAP Version 1.2 Specification Assertions and Test Collection (2003). http://www.w3.org/TR/2003/REC-soap12-testcollection-20030624/.

[10] Internet Engineering Task Force (IETF), HTTP Extensions for Distributed Authoring and Versioning (WebDAV), Request for Comments (RFC) 2518 (1999). http://www.ietf.org/rfc/rfc2518.txt.

[11] Satoshi Konno, CyberLink for Java. http://www.cybergarage.org/net/upnp/java/index.html.

[12] IEEE, Standard for High Performance Serial Bus, IEEE Std 1394-1995 (1995).

[13] IEEE, Standard for High Performance Serial Bus Amendment 1, IEEE std 1394a-2000 (2000).

[14] IEEE, Standard for High Performance Serial Bus Amendment 2, IEEE std 1394b (2002).

[15] Internet Engineering Task Force (IETF), IPv4 over IEEE 1394, Request for Comments (RFC) 2734 (1999). http://www.ietf.org/rfc/rfc2734.txt.