DESIGN AND EVALUATION OF THE HOME NETWORK SYSTEMS USING THE SERVICE ORIENTED ARCHITECTURE

Hiroshi Igaki, Masahide Nakamura, Ken-ichi Matsumoto
Graduate School of Imformation Science, Nara Institute of Sciense and Technology, 8916-5 Takayama-cho, Ikoma, Nara Japan

Keywords: Web Services, Service-oriented architecture, Home network, distributed system

Abstract: In the conventional home network systems (HNS), a powerful centralized server controls all electric home appliances connected to provide value-added integrated services. However, when the number of the appliances increases and the appliances become more sophisticated, the conventional architecture would suffer from problems in superfluous resources, flexibility, scalability and reliability. This paper proposes alternative architecture for HNS, which exploits the service-oriented architecture with Web Services. In the proposed architecture, each appliance is controlled by a Web service in a de-centralized manner. Then, the services autonomously collaborate with each other to achieve the integrated service scenarios. To evaluate the HNS at the design process, we also present four kinds of evaluation metrics: reliability, load, complexity, and coupling. Using these metrics, we conduct a comparative study among the proposed and the previous HNS architectures.

1 INTRODUCTION

Recent advancement in computer network technology enables electric home appliances to be connected in a network. The appliances, such as an air-conditioner, door sensors, lights, a TV and a DVD player, are connected with each other. The system consisting of such networked home appliances is generally called a *Home Network System* (HNS for short). Several commercial HNS products are already on the market (e.g., LG E, 2004; Samsung, 2004; Hitachi, 2004).

The appliances in HNS are controlled together to provide *integrated services*, which add more value and convenience to the daily life of home users. Typical integrated services include;

If the user comes home, the lights and the air-conditioner are automatically turned on.

When the user starts to watch DVD movies, the lights becomes dark and the volume on the TV is adjusted.

The current HNS mainly adopts the *server centralized architecture* (we call it SCA in the following), where a powerful and intelligent server (called *Home Server*) controls all the dumb appliances connected. In general, each appliance does not have advanced intelligence, and it just receives (a sequence of) commands from the server with a low-level and light-weight network adapter.

Since SCA is quite simple architecture, it is relatively easy to apply SCA to the HNS consisting of the *conventional* home electric appliances. However, in the near future, the SCA-based HNS will be faced with the following problems.

Since SCA generally requires proprietary middle-ware, it is difficult to achieve the interoperability among products from different vendors.

All the appliances heavily rely on the centralized Home Server. Therefore, the server suffers from the scalability problem when the number of appliances becomes large. Also, the server requires considerably high reliability, because all the integrated services stop when the server fails.

Even if the appliances come to have more intelligent processors and network devices, the HNS cannot make flexible use of the resources as the Home Server takes the main control. Thus, the quality of the integrated services is limited to the features implemented in the server.

This paper presents alternative architecture for HNS. Specifically, we propose to apply the *service oriented architecture* (SOA, for short) (Hao, 2003) with Web services to HNS. SOA is basically

Igaki H., Nakamura M. and Matsumoto K. (2004).
DESIGN AND EVALUATION OF THE HOME NETWORK SYSTEMS USING THE SERVICE ORIENTED ARCHITECTURE.
In *Proceedings of the First International Conference on E-Business and Telecommunication Networks*, pages 62-69
DOI: 10.5220/0001398300620069

architecture to integrate distributed self-contained services using loose coupling and well-defined interfaces. In this paper, we assume the next-generation home electric appliances, which are intelligent enough to process Web service transactions with own processors and network devices.

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

In this paper, we conduct the architectural design of a practical HNS example using the SOA framework. Then, we propose a graph-based method to evaluate the design quantitatively, from the viewpoints of reliability, workload, functional complexity and coupling. These methods are applied to two different HNSs with SOA and SCA, in order to see the difference.

2 SERVICE-ORIENTED ARCHITECTURE (SOA) AND WEB SERVICES

SOA is an architectural style whose goal is to achieve loose coupling among interacting autonomous software agents. A *service* is a unit of tasks done by a service provider to achieve desired end results for a service consumer. The *interface* of a service is strictly typed so as to be processable by software agents of the service consumers. Through the interface, features of the service are exported to the network as methods. Since the interface is supposed to be unchanged, the consumer can use the service from a remote place, as if it were just an ordinary method invocation, without knowing internal logic or protocol message formats. This is known as the *loose coupling*. Using this concept, a service can autonomously collaborate with other services, which enables more sophisticated integrated services.

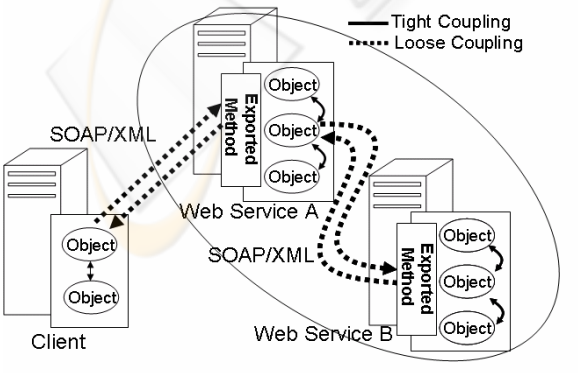


Figure 1: Service-Oriented Architecture

A Web service (Ethan, 2002; W3C, 2004) provides an open and standard means to implement the SOA-based system. The interface of a Web service is described by XML-based format, specifically WSDL. Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

Figure 1 shows an example of SOA using Web service. The client application (Client) accesses the first Web Service A through its exported method. Web Service A internally calls a method of Web Service B. Web Service B returns the result to Web Service A, and finally Client gets the end result. The interface of the exported methods is described by WSDL, and Client and Web Services are loosely coupled by SOAP/XML. As a result, Client uses the integrated service consisting of Web Service A and Web Service B (depicted by a large oval in the figure).

3 DESIGNING HOME NETWORK SYSTEM (HNS) WITH SOA

3.1 Key Idea

Considering today's evolution of network technology, it is reasonable to assume that the next-generation home electric appliances can be autonomous nodes with software control, supported by own processors and network devices (DHWG, 2004).

Our key idea is to apply SOA to such autonomous home electric appliances. Specifically, each appliance has a software layer (we call it *service layer*) from which its end device (hardware) can be controlled. Then, we implement an interface of the control in the service layer as a Web service, and export it to the network. By doing this, multiple appliances can autonomously collaborate with each other at the service layer. This enables to develop more interoperable and flexible integrated HNS services.

For instance, suppose that a Web service of room lights provides "SwitchON" method to the network. Then, a door sensor can collaborate with the lights by executing the method, so that the lights are turned on when the user opens the door. Note that this integrated service does not require any centralized server. Also, the communication between the sensor and the lights is done in terms of a standardized manner of Web services.
In the following subsections, we demonstrate how a practical HNS can be designed based on the proposed architecture with SOA and Web services.

3.2 Target Home Network System

As a practical example, in this paper, we try to design an HNS consisting of the following 9 home electric appliances: a DVD player, a TV, a speaker, a light, an illuminometer, a door, a telephone, an air-conditioner and a thermometer. In this HNS, we achieve the following eight service scenarios (denoted by SS) as the integrated services. These scenarios are taken from actual commercial products (ECHONET, 2004; Samsung, 2004).
SS1: The brightness of the light is automatically adjusted based on the current intensity of illumination with the illuminometer.
SS2: If the user enters a room from the door, the light are turned on.
SS3: When the user turns on the DVD player, the light becomes dark. Then, the TV and the speaker start in the DVD mode.
SS4: When the user watches the TV, the speaker is turned on.
SS5: While the user is watching the TV, if the telephone rings, then the volume of the speaker becomes small.
SS6: The air-conditioning is optimized based on the thermometer.
SS7: If the user enters the room, the air-conditioner starts and adjusts the temperature to a comfortable degree.
SS8: When the user goes out or goes to bed, all the appliances are shut down and the door is securely locked up.

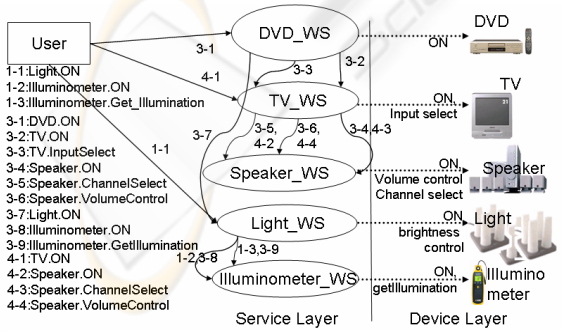


Figure 2: An HNS with SOA (containing SS1, SS3, SS4)

3.3 Design of the HNS with SOA

As discussed in Section 3.1, we assume that each appliance is an autonomous intelligent node, which can control the end device by the software. Also, each appliance is supposed to have enough processing power to operate own Web service to export its control to the network.
With the assumption, we here try to conduct an architectural design of the target HNS in Section 3.2 with SOA. Specifically, we consider what methods must be implemented in the Web service (denoted by WS, in the following) of each appliance, in order to achieve all the service scenarios SS1 to SS8 in the target HNS.
Figure 2 shows an architectural design involving a part of the service scenarios (SS1, SS3 and SS4). Each appliance consists of an end device and the corresponding WS (depicted by an oval). The whole architecture is divided into two layers: the *device layer* and the *service layer*. In the device layer, an end device is controlled directly by the corresponding WS (drawn by a dotted line). On the other hand, in the service layer, features of each appliance are exported as methods of the corresponding WS.
To provide the integrated service scenarios, the appliances collaborate with each other via the network, by autonomously executing the exported methods. In Figure 2, a solid arrow with label L from WS A to B means that WS A executes (uses) the method L provided by B. Due to the limited space, each label is represented by a number in the form of i-j describing j-th method executed in SS_i (i=1,3,4).

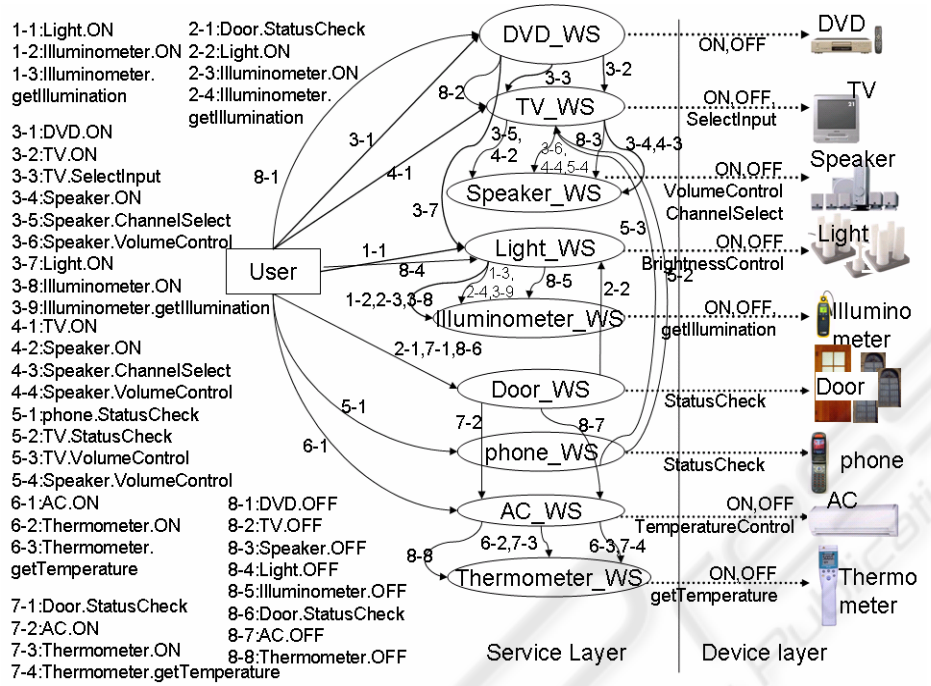


Figure 3: An HNS with SOA

Let us consider SS1 in Section 3.2. In Figure 2, we can see a possible design to implement SS1, by traversing arrows prefixed by "1-". First, the user calls the method Light.ON to Light_WS. Next, Light_WS turns on the illuminometer with Illuminometer.ON, and acquires the current illumination by Illuminometer.GetIllumination. Finally, Light_WS sets up the optimal lighting to the lighting devices based on the present illumination. Similarly, by traversing the arrows prefixed by "4-", we can see the scenario SS4, where the TV autonomously turns on the speaker and adjusts the speaker volume.

SS3 can be achieved by reusing SS1 and SS4. The user first turns on the DVD player by DVD.ON. Next, DVD_WS executes TV.ON and TV.InputSelect for TV_WS, and calls Light.ON for Light_WS. Then, Light_WS and TV_WS execute the existing SS1 and SS4, respectively. Thus, the SOA allows us to reuse and integrate the existing scenarios to achieve a new integrated service.

An architectural HNS design containing all the scenarios SS1 to SS8 is shown in Figure 3

The main characteristics of the SOA-based HNS are summarized as follows. The distributed appliances collaborate with each other to realize the integrated service scenarios on demand from the user. Each WS can be used as a reusable component to construct integrated service scenarios. Since the control of HNS is fully distributed, the implementation of each WS is expected to be simple and self-contained.

3.4 Design of the HNS with SCA

For the comparison purpose, we also consider the target HNS with the server centralized architecture (SCA), which is adopted by most of the current commercial HNS products (Hitachi, 2004; Samsung, 2004). In these products, a light-weight adapter is connected to the conventional home electric appliance. The adapter relays commands from the powerful centralized server, called Home Server (HS, for short). The communication is performed by the proprietary software and protocol.

Figure 4 shows the architectural design of HNS with SCA. In this HNS, all the appliances are directly controlled by the HS. The integrated service scenarios are performed by cooperation of (tightly-coupled) objects in the HS. For example, when the user demands to execute SS3, Home Server directly sends the proprietary commands to the DVD player, the TV, the speaker and the lights.

Thus, in the SCA-based HNS, the architecture itself is quite simple since HS takes the control of all appliances. However, the implementation of HS tends to be complex, and the workload of the HNS is concentrated in HS. Also, all the integrated services become unavailable if HS fails. These issues are discussed quantitatively in the next section.

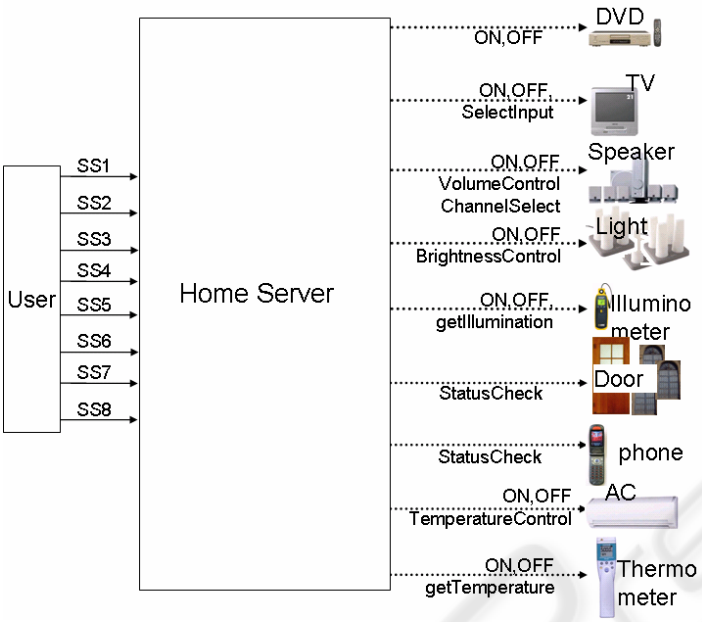


Figure 4: An HNS with SCA

4 EVALUATION OF HNS ARCHITECTURAL DESIGN

In this section, we propose a graph-based method to perform quantitative evaluation of the HNS architectural design. For a given HNS design, the proposed method derives four kinds of metrics: reliability, workload, functional complexity and coupling.

4.1 Service Integration Graph

As seen in Figure 2, Fig. 3 and Figure 4, an HNS with integrated service scenarios (we simply call *scenarios* in the following) can be characterized by a labelled directed graph, where a node represents an HNS component (i.e., a user, an end device, a WS or an HS), and a directed edge denotes a method invocation among the components. By utilizing the graph, several important characteristics of the HNS can be mathematically derived.

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

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

Next, we extend the service integration graph to the *set* of scenarios. Let $s_1, s_2,...,s_k$ be a given set of scenarios. For each i ($1 \le i \le k$), we have $SIG(S_i)=(Ns_i,Ls_i,Es_i)$. Then, we define $SIG(s_1, s_2,\ldots, s_k)=(\cup_i Ns_i, \cup_i Ls_i, \cup_i Es_i)$. If $s_1,s_2,...,s_n$ are all the scenarios in the HNS, then we call $SIG(\{s_1, s_2,\ldots, s_n\})$ a *full service integration graph*, which is denoted by *FSIG*. Note that for a given HNS, any *SIG* is a subgraph of *FSIG*.

For instance, consider the scenarios SS1 to SS8 in Section 3.2. We can see that Figure 2 represents $SIG(\{SS1,SS3,SS4\})$ and that Fig. 3 represents $FSIG$ $(=SIG(\{SS1,SS2,...,SS8\}))$.

4.2 Reliability

Assuming that each HNS component may fail, we evaluate the system-wide reliability of HNS from a viewpoint of the availability of the integrated services. For a given HNS with scenarios, we define *n-reliability* as the probability that at least n scenarios are available in the HNS. The n-reliability varies depending on the architecture as well as the reliability of each component. Evaluating the reliability at the design process is crucial for reliable system implementation.

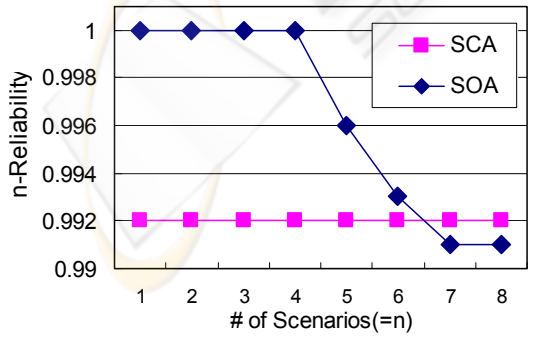
To calculate n-reliability, we apply the Sum of Disjoint Products (SDP) approach (Hariri, 1987; Soh, 1991; Tsuchiya, 2000) to the service integration graph. The SDP is a method to derive the network reliability based on pathset and cutset of the graph theory. Intuitively, when a graph G and reliability of each node (and edge) are given, the SDP method

calculates reliability that at least one of specified set of subgraphs of *G* is available (i.e., *operational*), by taking the overlaps among the subgraphs into account.

As seen in the previous subsection, each scenario in HNS is characterized by a *SIG*, and a *SIG* is a subgraph of *FSIG*. Hence, n-reliability can be calculated by SDP in such a way that some n *SIGs* are operational in *FSIG*. For instance, in our target HNS, 1-reliability is calculated by SDP as a probability at least one of *SIG*(SS1), ..., *SIG*(SS8) is operational. Similarly, 2-reliability is derived from *SIG*({SS1,SS2}),*SIG*({SS1,SS3}),...,*SIG*({SS7,SS8}). Thus, taking all combinations from the given set of scenarios, we can compute n-reliability with the SDP method.

To evaluate the reliability purely relevant to the architecture of HNS, we assume that only WS (in SOA) and HS (in SCA) may fail. As an expected value, we set the reliability of each WS (in SOA) to be 0.999. We also set the reliability of the HS (in SCA) to be 0.992 (=0.999^8, since HS implements proprietary programs for 8 scenarios). Then, we applied the SDP method to the SOA-based HNS (in Fig. 3) and the SCA-based HNS (in Figure 4).

The result is shown in Figure 5. The horizontal axis represents the number of scenarios (n), while the vertical axis plots n-reliability. From the result, it can be seen that n-reliability for SCA becomes equal to the reliability of HS. This is because all scenarios depend on the centralized HS. In other words, if the HS fails, all the scenarios become unavailable. On the other hand in SOA, the eight scenarios use distributed WS. Hence, even if a WS crashes, scenarios are partially operational. Thus, the SOA-based HNS achieves higher fault tolerance than the SCA-based HNS. For n=7,8, SCA achieves slightly more reliable than SOA. This is because the probability that all the components in SOA are operational becomes smaller than that of SCA, since SOA contains more components.



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

Figure 5: Reliability

Table 1: Workload

(a)SOA-based HNS

WS	WL(WS)
DVD_WS	10.7
TV_WS	29.8
Speaker_WS	29.8
Light_WS	57.4
Illumino_WS	57.4
Door_WS	18.7
Phone_WS	3.7
AC_WS	16
Thermo_WS	16
StandardDev	18.203

(b)SCA-based HNS

HomeServer	WL(HS)
HS	86.2
StandardDev	86.2

4.3 Workload

Our interest here is to measure a workload of each component (WS or HS) imposed when performing integrated services in HNS. The workload varies depending on the *usage frequency* of scenarios. Based on the given usage frequency, we characterize the workload of each component *v* as a total number of appearance of *v* in all scenarios. This metric enables us to determine the deviation of workload in HNS, so that we can change the design of HNS in consideration of load-balancing.

Suppose that we have *FSIG*=(*N*,*L*,*E*) and scenarios $s_1,...,s_n$. Also suppose that f_i ($1 \le i \le n$) is a given usage frequency of scenario s_i. For each node $v \in N$, we define an *appearance function* $c_i: N \rightarrow \{0,1\}$ such that: $c_i(v) = 1$ iff *v* appears in $SIG(s_i)$, otherwise $c_i(v)=0$. Then, a workload for the component *v* is defined by $WL(v) = \sum_{i=1}^{n} f_i \times c_i(v)$

For the evaluation of our target HNS, we interviewed 12 users (8 singles, 2 married men without children, 2 men with a family of four). We asked them the estimated usage frequency of the scenarios SS1 to SS8 per week, and obtained the average number of usage of each scenario. Based on this, we calculate the workloads of WS (in SOA) and HS (in SCA).

The result of the workload estimation is shown in Table 1. The column *WL*(WS) shows how many times each WS (or HS) is used per week. The result for SOA gives important information on which components require load-balancing. For example, since *WL*(Light_WS) is large, it would be reasonable to prepare a backup WS to share the load. Thus, in the SOA-based HNS, it is relatively easy to perform flexible design changes reflecting the workload. On the other hand, from the result of SCA, we can see that HS suffers from much heavier workload than those of SOA. The only way to perform the load-

balancing is duplicate the HS, which is not as flexible as the case of SOA.

4.4 Functional Complexity

In this subsection, we estimate the functional complexity for each component at the design stage. Basically, the functional complexity for a component v depends on how many methods v has to *provide* and *use*, in order to achieve all the integrated services. This is a key factor for implementing v. Specifically, for each node v in FSIG, we count the number of the labels attached to the incident edges of v.

Let $FSIG=(N,L,E)$ be given. An edge $(WS_A,m,WS_B)\in E$ describes that WS_A uses the method m of WS_B by definition. So, the function of m should be implemented inside WS_B. In this sense, we call m internal function of WS_B. On the other hand, from the viewpoint of WS_A, WS_A has to call m which is outside WS_A. Hence, m is called external function of WS_A.

For each component $v\in N$ in *FSIG*, we define the functional complexity of v as the number of internal and external functions of v. Strictly speaking, the number of internal functions of v is defined as $inum(v) = |\{m|\exists v';(v',m,v)\in E\}|$. Also, the number of external functions is defined as $enum(v) = |\{m|\exists v';(v,m,v')\in E\}|$. Then, the functional complexity of v is defined by $fcomp(v) = inum(v) + enum(v)$.

For example, let us take Light_WS in Figure 3. Then,
inum(Light_WS) = |{1-1:Light.ON, 2-2:Light.ON, 3-7:Light.ON, 8-4:Light.OFF}| = 2
enum(Light_WS) = |{1-2:Illuminometer.ON, 1-3: Illuminometer.getIllumination, 2-3: Illumino-meter.ON, 2-4: Illuminometer.getIllumination, 3-8: Illuminometer.ON, 3-9: Illuminometer.getIllumina-tion, 8-5: Illuminometer.OFF, LightON, LightOFF, LightBrightnessControl}| = 6

Table 2 shows the functional complexity for all the components of our target HNS. It can be seen that each WS in SOA requires a smaller number of functions than HS in SCA. This implies that the effort taken for the implementation of WS would be smaller than that of SCA. Also for the SOA-based HNS, it is also possible for the designer to make the functional complexity well-balanced, by carefully modifying the scenario design (i.e., changing the topology of *FSIG*).

4.5 Coupling

The coupling measures the degree of dependence of a component against other components. Although the coupling between WS (in SOA) is basically loose (see Section 2), it provides a reasonable guideline for robust scenario designs. If a WS v is used by (or uses) a lot of other components, failure of v affects these components, which dramatically decreases availability of the service scenarios.

Table 2: Complexity and Coupling

	Complexity		Coupling	
WS/HS	inum(WS)	enum(WS)	use(WS)	used(WS)
DVD_WS	2	6	3	1
TV_WS	5	7	2	3
Speaker_WS	4	4	1	1
Light_WS	2	6	2	3
Illumino_WS	3	3	1	1
Door_WS	1	4	3	1
Phone_WS	1	2	2	1
AC_WS	2	6	2	2
Thermo_WS	3	3	1	1
HS	8	23	9	1

Let $FSIG=(N,L,E)$ be given. For each component $v\in N$, we define *coupling* of v as the total number of components that v uses or are used by v. Strictly speaking, for $v\in N$, let $use(v) = |\{v'|\exists m;(v,m,v')\in E\}|$ and $used(v) = |\{v'|\exists m;(v',m,v)\in E\}|$. Then, coupling of v is defined by $coup(v) = use(v) + used(v)$.

For example, let us take TV_WS in Figure 3. Then,
use(TV_WS) =|{speaker_WS, TV}|=2
used(TV_WS) =|{a user, DVD_WS, telephone_WS} |=3. Hence, *coup*(TV_WS) = 5.

The coupling for all the components of our target HNS is shown in right-half of Table 2. It can be seen in that the coupling of all WS (in SOA) is well-balanced. We can also see that the components in SCA are heavily dependent on the HS. This implies that the crash of HS is fatal, which is as discussed in Section 4.2.

5 DISCUSSION AND CONCLUDING REMARKS

In this paper, we proposed an application of the service-oriented architecture to HNS. We also presented a graph-based method to evaluate the architectural design of HNS. With a case study, we evaluated the HNS design using the four kinds of metrics and discussed the difference between SOA and SCA, quantitatively.

Of course, there are other important factors that we could not cover in this paper; such like performance, security, and implementation issues, etc. Therefore, we cannot say that the proposed SOA-based HNS is absolutely superior to the conventional SCA-based HNS. Instead, our contribution is to show the applicability of SOA to the HNS through quantitative evaluation.

In the market of home electric appliances, a shift from dumb appliances to intelligent appliances is imminent. More convenient and more sophisticated integrated services will be required in the HNS such as entrance management and apparatus operation with user's voice. To make full use of such intelligent appliances in the HNS, SOA is quite promising architecture, as shown in this paper.

Another contribution is to present the concrete evaluation method for the architectural design of HNS. The proposed metrics provide useful information for the HNS developers to make various decisions on design, implementation and usability of HNS, at the early stage of the development.

Some topics for future research present themselves. We are currently implementing an HNS simulator with Web services in a distributed environment. The more practical evaluation using the simulator would allow us to find other useful metrics for the HNS development. In multi-user HNS, the *feature interaction* problem (Michael, 2003) must be considered, which is known as a functional conflict among scenarios and/or appliances. Investigating practical solution of the feature interaction problem is also our future work.

ACKNOWLEDGEMENT

This work is partly supported by Grand-in-Aid for COE (Center Of Excellence) and Encouragement of Young Scientists (No.15700058), from Research of the Ministry of Education, Science, Sports and Culture, Japan.

REFERENCES

DHWG, (2004) Digital Home Working Group, [Online], Available: http://www.dhwg.org/ [2004].

ECHONET, (2004) ECHONET CONSORTIUM, [ONLINE],Available: http://www.echonet.gr.jp/english/ index.htm [2004].

Ethan, C., (2002) Web Services Essentials, United Stated of America: O'Reilly & Associates, Inc.

Hitachi. (2004) Horaso Network Service, [ONLINE], Available: http://ns.horaso.com/ [2004].

LG E., (2004) Home Network, [ONLINE], Available: http://www.lge.com/products/homenetwork/homenetwork.jsp [2004].

Michael, W., (2003) 'Feature Interactions in Web Services', Proc. of Seventh Int'l. Workshop on Feature Interactions in Telecommunication Networks and Distributed Systems (FIW'03), pp.149-156.

Samsung. (2004) Home Network, [ONLINE], Available:http://www.samsung.com/HomeNetwork/index.htm [2004].

S. Hariri, and C. S. Raghavendra, (1987) 'SYREL: A Symbolic Reliability Algorithm Based on Path and Cutset Methods', IEEE Transactions on Computers, October, pp.1224-1232.

Soh, S. and Rai, S., (1991) 'CAREL: Computer aided reliability evaluator for distributed computing networks', IEEE Trans. Parallel and Distributed Systems, July, pp.199-213.

T., Tsuchiya, T., Kajikawa, and T., Kikuno, (2000) 'Parallelizing SDP (Sum of Disjoint Products) Algorithms for Fast Reliability Analysis', IEICE Transactions on Information and Systems, Vol.E83-D, No.5, May , pp.1183-1186.

W3C. (2004) W3C Web Service Activity, [ONLINE], Available: http://www.w3.org/2002/ws/ [13 Feb 2004].

Hao, H., (2003) What is Service-Oriented Architecture?, [ONLINE], Available: http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html [30 Sep 2003].