A Visual Framework for Monitoring and Controlling Distributed Service Components

Takahiro Kimura†,　Haruaki Tamada†,　Hiroshi Igaki†,
Masahide Nakamura†　and　Ken-ichi Matsumoto†

† Graduate School of Information Science, Nara Institute of Science and Technology
{taka-kim, harua-t, hiro-iga, masa-n, matumoto}@is.naist.jp

Abstract　The emerging ubiquitous networked technologies enable to integrate distributed software/hardware components to achieve more value-added services.　In order to develop and manage such integrated services effectively, it is essential to monitor the status of each component during runtime, preferably with an intuitive and visual appeal.　In this paper, we propose a component monitoring framework that achieves the following three requirements: (R1) monitoring status of components, (R2) control of the components and the integrated services,　and (R3) visualization of component's states. Based on the proposed framework, we have implemented a monitoring system of a home network system, where ten home appliances are orchestrated to provide value-added services. For the implementation, Flash and Java Servlet have been extensively used.

Keywords　distributed service component, monitoring, visualization, control, framework, Service Oriented Architecture

1. Introduction

The advancement of network and software technologies enables the rapid and flexible development of software systems using *distributed software components*.　Various features of a system are implemented as reusable components, which can be executed remotely via the network using technologies like RMI[3], CORBA[2] and Web Services[7].

Recently in the area of the software components, the concept of *Service-Oriented Architecture (SOA)*[6] has attracted a public attention, and the Web Service [7] is widely known as an infrastructure to achieve the SOA.　In the SOA, the system features are aggregated into a set of coarse-grained *service*. Each service is implemented as a self-contained service component which does not depend on the context or state of the other services.　The service components distributed over the network can be loosely coupled with the platform-independent interfaces. This enables the rapid development of more *value-added services*. Since the SOA facilitates the orchestration of *heterogeneous distributed systems*, the application to the enterprise systems has been widely discussed.

We believe that the SOA fits well with the emerging ubiquitous environment, where various heterogeneous hardware devices and software applications are networked to provide value-added services.　Making the features through service components, the devices and applications can work together without having the implementation-specific problems.　In [5], we proposed a method to apply the SOA to a home network system.

Basically, the SOA poses a principle that each service component should be *stateless*, in the sense that a service does not care the state of other services. However, each service component actually has a state if the component wraps a *stateful* object.　For instance, a service component for controlling a DVD player may behave differently depending on the state of the DVD like; standby, playing and forwarding.

Thus, when we develop value-added services by integrating service components involving stateful objects, it is essential to monitor the state of the components. Moreover, it is preferable to visualize the state during runtime in an intuitive manner. Indeed, there exist several tools for monitoring service components within a SOA framework [8].　However, the tools basically monitor the performance and data of workflow instead of the state of components.

The goal of this paper is to present a framework that can be used to monitor and visualize the state of service components, as well as to trigger the component-integrated services.　Specifically, we first define three requirements for the framework, where the system must be able to

(R1) acquire the current state of the underlying object during runtime, and
(R2) visualize the current state of the all components involved in the integrated service,
(R3) allow users to invoke the integrated services.

Then, we have designed the framework based on the above requirements. For (R1) we embed an API for polling the state via Java Servlet. Specifically, for (R2) we employ Flash interface to visualize the component states. To achieve (R3), we connect the Flash interface with the components via Java Servlet.

Using the proposed framework, we have implemented a status monitoring system of a home network system. The system consists of ten home appliances, each of which is wrapped by a service component. We have developed eight kinds of value-added services by integrating the components.　The monitoring system successfully helped us to capture intuitively, how the integrated services work and change the state of each appliance.

2. Preliminaries

2.1. Service Oriented Architecture

The service oriented architecture (SOA) [6] is a system architecture to integrate different systems distributed over a network with a standard procedure. Each system exports its own features as *services*. (a set of tasks, which are coarser than objects). The internal logic and implementation of a service are self-contained and encapsulated in the system. The system exposes only interfaces of the service in the form of strictly-typed *exported methods*.

A service user executes the remote exported method and gets desired results. This remote procedure call is performed by a standardized platform-independent framework (like XML/SOAP).

Also, once an exported method is deployed, its interface definition is not allowed to change. Therefore, the changes in the internal service logic or service implementation platform do not influence the service user. Thus, the loose coupling between the user and the service is achieved. *Web Services* [7] are widely known as a major SOA framework.

2.2. Service Component and Integrated Services

Once a system deploys services based on SOA, the system can be regarded as a reusable software component, which we call *service component*. Due to the nature of the SOA, an external program can access a service component using a standard interface, without knowing the internal logic or implementation platform of the service component.

Figure 1 shows an example service component for a DVD recorder. This component consists of three layers: (a) a hardware layer of the DVD recorder, (b) a software layer comprising embedded objects for controlling the hardware, and (c) a service layer which exposes the features of the recorder as services by aggregating some objects. In this example, three services are exported to the network: Power, Playing, Recording. These services are supposed to be accessible from network via a certain standard method. In this paper, we assume Web service [7] for the service access method.

We can use a single service component for itself. However, when there are other service components available over the network (called *distributed service components*), combining multiple components together yields value-added *integrated services*. For instance, if we have service components for TV, lights and speakers, then we integrate them with the DVD recorder to implement *a DVD theater service*, where the user can watch DVD in a theater-like atmosphere.

2.3. Developing Integrated Services

In order to implement the integrated service from multiple service components, it is necessary to define how the integrated service executes the individual services provided by components.

To integrate distributed service components, a framework called BPEL4WS [1] has been standardized. BPEL4WS integrates multiple Web Services as a business workflow. BPEL4WS regards each invocation of a Web service or data manipulation as an activity, and defines the execution method among activities (e.g., parallel execution, sequential execution, etc.). Recently, several support tools to develop reliable integrated services come onto market. IBM's WebSphere Business Integration Server Foundation [8] includes a set of support tools to develop

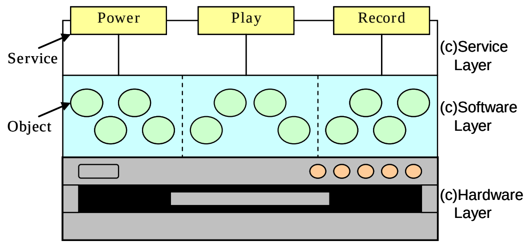

Fig.1: Example of SOA by DVD Recorder

BPEL-based integrated services. "WBI Modeler" visualizes the workflow to help the developer to define the relationships among activities. "WBI monitor" monitors the runtime behavior of the workflow for performance measurement and message logging.

2.4. Monitoring Distributed Service Components

BPEL4WS deals with the total workflow of the integrated services, and it does not care details of individual service components. Thus, the existing support tools, WBI Modeler and WBI monitor, focus only how the workflow is executed, and monitoring the internal status of individual service components is out of scope.

However, from the viewpoint of service developer, it would be more desirable to see intuitively how each service component behaves when executing the integrated service. This is especially crucial when the service component controls the underlying stateful object (like electric appliance) and the object is not within a directly visible distance from the developer.

Our goal is to support the development of the integrated services with distributed service components more efficiently. To achieve the goal, this paper proposes a framework to monitor the state of each service components during runtime.

3. Framework Requirements

In this section, we present three requirements for the proposed framework. Specifically, the requirements are: (R1) Monitoring states of service components, (R2) Visualizing states of service components, and (R3) Invoking integrated services and service components. These requirements are intended to be implemented in any system to monitor the status of the distributed service components.

3.1. Monitoring states of service components

When developing integrated services using distributed service components, only what the developer can directly see is the interface of the services. The developer cannot (or needs not to) see the internal logic, dependencies integration logic of the service components. In order for the developer to check if the components are working correctly, a reasonable way is to get the *state* of the component during runtime.

In general, each service component has a set of principal *properties* that characterizes the behavior of the component. For instance, the DVD recorder would have properties `Power` (ON or OFF) and `DriveMode` (Idle, Playing or Forwarding). These principal properties dynamically vary during runtime according to the execution of integrated services, the direct operation of the service.

So, we define a (current) state of a service component as a snapshot of values of all the principal properties. As the first requirement of the proposed framework, a monitoring system must have a feature to acquire the latest current state (i.e., the list of property values) of each service component. Preferably, the state should be obtained periodically and autonomously, regardless of the user's action. Also, the state should be represented in an implementation independent format (like plain text or XML).

3.2. Visualizing state of service components

As stated in the previous requirement, a state of a service component is obtained in the implementation independent data. The next requirement is to *visualize* the state in an intuitive and clear representation for the developer. The visualization significantly helps the developer to check visually if each component behaves as expected, during runtime of the integrated services.

Preferably, all of the visualized components should be displayed in one screen, so that the developer can see all the distributed service components at once.

3.3. Invoking integrated services and service components

In addition to monitoring visually the distributed service components, the developer should be able to invoke integrated services or to operate directly some components from the monitoring system. This is essential for debugging and testing the integrated services efficiently.

The integrated services are often managed by an external service manager (e.g., a BPEL engine), which orchestrates a set of service components necessary for each integrated service. Therefore, the monitoring system must have a feature to send a request to the service manager, so that the service manager triggers the integrated service request. For this, the monitoring system should send a trigger only, and should not interfere on the execution logic, integration scheme, or the context of the integrated services.

On the other hand, the direct operation to each service component is performed by executing an exported method of the component. Thus, upon receiving a request from the developer, the monitoring system should be able to dynamically invoke the exported method according to the pre-defined service interface.

4. Design of Framework

Figure 2 shows an overview of the proposed framework. The boxes in the bottom of the figure represent service components. These service components are executed by an integrated service manager is in the right-middle of the figure. The proposed framework mainly consists of two parts: the Flash Interface and Java Servlet, which are surrounded by the dotted oval.

4.1. Flash Interface

In order to visualize the current state of the service component (Requirement R2 in Section 3), we extensively utilize *Flash* as a user interface. Flash is a tool kit developed by Macromedia for flexible creation of Web contents involving animation. Using Flash, we represent each state of a service component by a Flash animation, which can allow a user to capture the component in an intuitive and visual manner. To get the current state of a service component, the flash interface has to access the service component autonomously during runtime.

For this, we use *Action Script*, which is a language to control Flash animations. In the proposed framework, we associate each event (e.g., mouse click or button press) on the Flash interface with an action script that accesses a service component. In fact, the communications between the Flash interface and service components are performed via Java Servlet in our framework. This is due to the security design of Flash toolkit, as well as the loose-coupling among the Flash interface and service components, which will be explained in the next section.

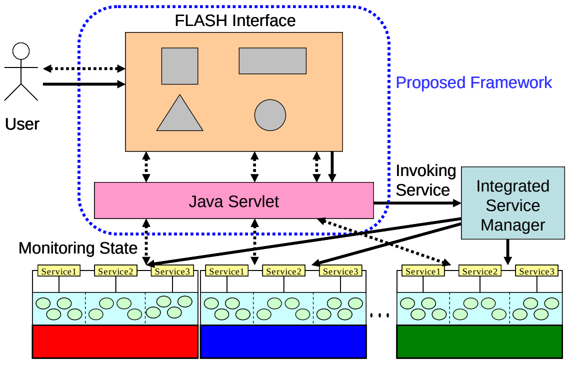

Fig.2: Overview of the proposed framework

4.2. Monitoring Service Components with Java Servlet

To monitor the current state of each service component (Requirement R1 in Section 3), we use Java Servlet. In order for the user to monitor the state of a service component, we need to first obtain the current state from the component, and then display the state in the Flash interface. However, due to the security reason, any Flash application is, by design, not allowed to access to applications in external servers. This design issue is critical for our framework. Therefore, we let Java Servlet to play a role of *proxy*. As depicted by dotted arrows in Figure 2, the Flash interface (with the action scripts) periodically sends a request to the Java Servlet. Then, the Java Servlet sends a query to the service components to obtain the latest states of the components. On receiving the current states, it returns the result to the Flash interface.

Introducing the Java Servlet as a proxy enables us to achieve *loose coupling* between the Flash interface and the service components. Any changes on the service components are managed in the Servlet. Also, the dependencies among service components are monitored by the Servlet. Therefore, the user and the Flash interface do not need to concern the changes or the relationships among components.

4.3. Executing Services with Java Servlet

To allow the user to execute integrated services or service components, we implement a mechanism of service invocation in the Java Servlet. Each operation to an integrated service or a service component is issued by the user through the Flash interface. Upon receiving a command, a corresponding action script of the Flash interface triggers the Java Servlet to execute services.

We implement two methods in the Servlet. The first method is to invoke the integrated service scenario. When the user selects a desired service scenario on the Flash interface, then the method triggers the integrated service scenario. For this, the method does not care which service components should be used. The method delegates the service control to the integrated service manager. The second method is to operate directly each service component. For each service component, we prepare primary interfaces of the component on the Flash interface. When the user trigger some action on the interface, the Flash interface passes the location of the service component and the name of the service to the Servlet as parameters. Then, the Servlet invokes the corresponding service of the component. The result of execution is reflected to the Flash interface according to the monitoring feature described in Section 4.2.

5. Application to Home Network System

We have applied the proposed framework to a home network system.

5.1. Home Network System

The Home Network System (HNS) is a system connecting home electric appliances, such as a TV, lights and air conditioner, to a local area network at home. Networking various appliances together can provide *value-added services*.

In [5], we proposed a framework to design and implement the home network system based on SOA (SOA-based HNS, for short). In the SOA-based HNS, each appliance is implemented as a service component consisting of two layers: service layer and device layer, as shown in Figure 1. In the device layer, the physical device is controlled by proprietary procedure or protocol. In the service layer, the appliance exposes device controlling interfaces as services to the network. The service layer also has a mechanism to trigger other appliances based on a given service scenario. Thus, the appliances can autonomously collaborate with each other on the service layer to provide integrated services.

In this paper, we prepare ten service components for the ten home networked appliances shown in Table 1. Each component has a set of exported methods (services) and a set of properties. For instance, the air conditioner has two methods: setPower() and setTemperature() for controlling power and temperature setting of the air conditioner. For simplicity, we just present two primary properties Power and TemperatureSetting which are modified by the execution of the methods.

Table.1: Example of Home Appliances and Properties

Home appliance	Method	Property

Air Conditioner	setPower()
	setTemperature()
		Power
		TemperatureSetting

Thermometer	setPower()
	setTemperature()
		Power
		CurrentTemperature

Speaker	setPower()
	setInput()
	setChannel()
	setVolume()
		Power
		Input
		Channel
		VolumeSetting

Light	setPower()
	setBrightness()
		Power
		BrightnessSetting

Illuminometer	setPower()
	getBrightness()
		Power
		CurrentBrightness

Door	getDoorStatus()	DoorStatus

Phone	ringing()
	connected()
		PhoneStatus

DVD player	setPower()	Power

TV	setPower()
	setInput()
		Power
		Input

Blind	setPower()
	setGate()
		Power
		BlindStatus

Table.2: Example of service scenarios

Service ID	Service Name	Description
SS1	Auto-TV	The brightness of the light is automatically adjusted with the illuminometer based on the current intensity of illumination.
SS2	DVD Theater	If the user enters a room from the door with a door sensor, the light is turned on.
SS3	Coming Home Light	When the user turns on the DVD player, the light becomes dark. Then, the TV and the speaker start in the DVD mode.
SS4	Coming Home Air-Con	When the user watches the TV, the speaker is turned on.
SS5	Ringing And Mute	If the telephone rings while the user is watching the TV, then the volume of the speaker becomes lower.
SS6	Blind	The air-conditioning is optimized based on the thermometer.
SS7	Sleep	If the user enters the room, the air-conditioner starts and adjusts the temperature to a comfortable degree.
SS8	Auto -Illumination	When the user goes out or goes to bed, all the appliances are shut down, and the door is securely locked up.

We also prepared eight scenarios of integrated services as shown in Table 2. Each service is realized by integrating the service components together. For example, the Auto-TV service is implemented by orchestrating two components: TV and Speaker.

5.2. Home Network Simulator

For the ten service components and the eight integrated services, we have implemented a monitoring system based on the proposed framework, called *Home Network Monitor*.

5.2.1. Flash Interface

The Flash interface of the Home Network Monitor mainly consists of two primary features: appliance display and service scenario display. The appliance display provides a visual interface for each of the ten appliances (components). The interface shows a graphic which animates according to the current state of the component. The interface also has several GUI components for the user to operate the component directly. In the service scenario display, the eight scenarios of the integrated services are listed. The user can choose one of them and executes the selected scenario.

When the Flash interface is launched, it obtains current state (i.e., the current values of the properties) of each appliance via the Java Servlet. Then, the interface updates the appliance display based on the obtained states.
If the user executes an integrated service on the service scenario display, the flash interfaces sends the trigger to the Java Servlet.
When the user drives the service component directly through GUI components, the interface passes the name of the service and its parameter to the Java Servlet.

The result of the execution is reflected on the appliance display. Figure 3 shows a screen-shot of the flash interface of the home network monitor.

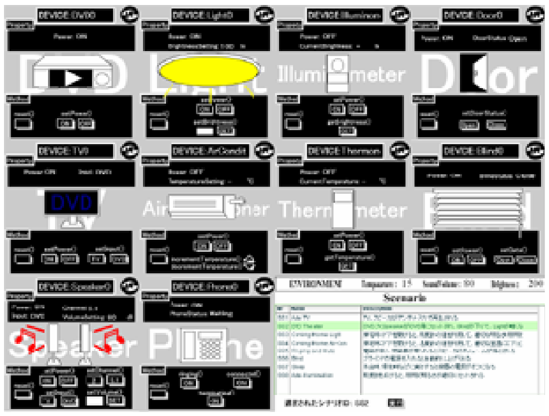
Fig.3: Screenshot of Home Network Simulator

5.2.2. Java Servlet

We have implemented three features in the Java Servlet of Home Network Monitor: (a) monitoring properties of home appliances, (b) updating properties of home appliances, (c) executing service scenarios. These three features were implemented within two kinds Java Servlets: Device Servlet and Invoke Servlet.

Figure 4 represents the component diagram of the Home Network Monitor. Table 3 represents a list of methods implemented in the two Servlets. The role of each method corresponds to one of the above features (a), (b) and (c).

Device Servlet provides a set of control methods to the Flash interface, which are related to monitoring and operating the service components. Specifically, Device Servlet provides three methods: getPropertyList(), getProperty(), and setProperty().

These methods get or set the current value of properties (thus, the current state) of each appliances. Upon receiving a request from the user, the Flash interface executes one of these methods. The result of the method invocation is returned in an implementation-independent format (i.e., plain text), so that the flash can understand the return value to update the animation.

On the other hand, Invoke Servlet offers two methods,

getAllDevices() and invokeService(), which are related to the execution of integrated service scenarios. Upon request from the user, the Flash interface executes invokeService() method. Then, Invoke Servlet triggers the scenario execution to the service component manager. The service component manager is an application module, which manages and executes the integrated services as well as service components (appliances), which is out of the Home Network Monitor.

Table.3: List of methods

Func tion	Method	Servlet	Description
(a)	getAll Devices()	Invoke Servlet	Obtain a list of home appliances deployed in the home network
	getProperty List()	Device Servlet	Obtain a list of properties for a home appliance
	getProperty()		Obtain a value of selected property
(b)	setProperty()	Device Servlet	Update a value of selected property
(c)	invoke Service()	Invoke Servlet	Execute a selected service scenario

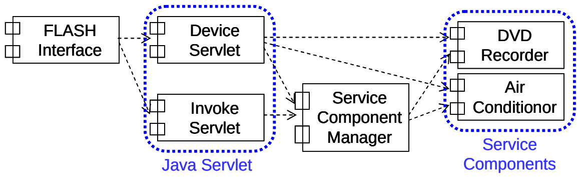

Fig. 4: Structure of SOA-based HNS

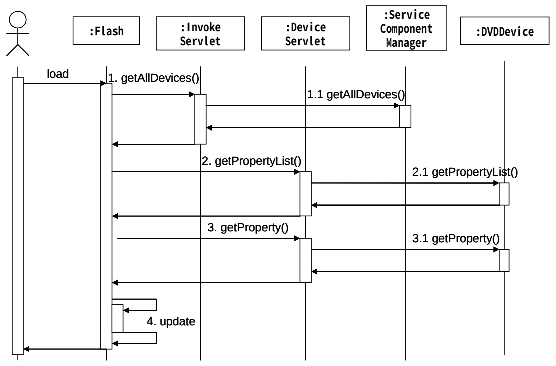

Fig. 5: Monitoring properties of home appliances

5.2.3. Monitoring properties of home appliances

Figure 5 represents a sequence diagram when the Home Network Monitor monitors the properties of appliances. The monitoring is performed according to the following four steps:

Step 1: Obtain a list of home appliances

When the user launches the Home Network Monitor, the Flash interface calls getAllDevices() method of InvokeServlet, and obtains a list of all appliances to be monitored.

Step 2: Obtain a list of properties

For each appliance obtained in Step 1, the Flash interface calls getPropertyList() method to gather all the property names of the appliance.

Step 3: Obtain the current state

For each property gathered in Step 2, the flash interface calls getProperty() method to obtain the current property values of each component.

Step 4: Display property values

Based on the property values obtained in Step 3, the Flash interface updates the appliance display. A property value is basically represented as a plain text. Hence, the value is reflected on Flash as a digit, string or an animation.

The Steps 3 and 4 are repeated periodically within a certain interval (usually, every few seconds), in order to continuously monitor the appliances.

5.2.4. Updating properties of home appliances

Figure 6 represents a sequence diagram when the user directly operates the appliance through the GUI components on the appliance display. The process is performed according to the following two steps:

Step 1: Get an event from user

When the user operates the GUI, the Flash interface gets the user's event. From the event, the Flash interface obtains the name of appliance, the service to execute, and parameters for the service.

Step 2: Execute the service

The Flash interface executes the requested service, and

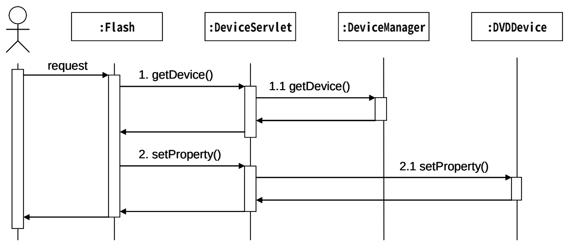

Fig. 6: Updating properties of home appliances

then sets the new property to the appliance by setProperty() method.

5.2.5. Executing service scenarios

Figure 7 represents a sequence diagram when the user executes an integrated service scenario on the Flash interface. The process is performed according to the following two steps:

Step 1: Get the choice of service scenario

When the user chooses and execute an integrated service scenario in service scenario display, the interface identifies which service scenario should be executed.

Step 2: Execute the service scenario

Based on the scenario selected, the Flash interface calls invokeService() method of InvokeServlet. The Servlet triggers the service component manager to initiate the integrated service requested. Once triggered, the integrated service is executed by the service component manager.

5.3. Discussion

From the experience in developing the Home Network Monitor, we summarize the effectiveness of the proposed framework. Introducing a monitoring application for distributed service components yields the following advantages:

- The user can simultaneously monitor all the service components involved in integrated services locally with one application.
- The monitoring system helps the user significantly to check if each service component works as intended.
- Using the monitoring system, the user can debug the scenarios of integrated services.

Based on the advantages, we consider the application of the proposed framework, with respect to the development stage and maintenance stage of service components and integrated services.

In the development stage, the developer of service components can use the monitoring application for *regression testing*, that is to verify service components under development (or modification). For this, the developer does not only confirm the component itself, but also check the *side-effect* of the component, which is an unexpected effect to other components caused by the modification.

In the maintenance stage, the operator can continuously monitor all the service components. If necessary, the operator can directly activate some components, or may send a maintenance command to the component. Also, the monitoring system helps the developer to inspire new integrated services.

Currently, the proposed framework does not address the creation of integrated services. In the maintenance stage, it will be necessary to create new integrated services, or modify the existing service scenarios. The issue of the service creation is left to our future work.

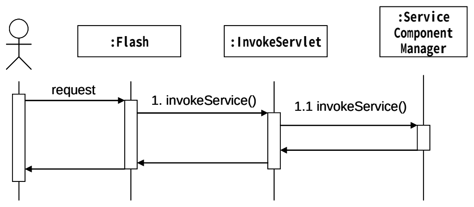

Fig. 7: Executing service scenarios

6. Conclusion

In this paper, we have presented a framework for monitoring distributed service components. The framework extensively uses Flash interface to visualize the state of the components, and employs Java Servlet to obtain the current state and to execute integrated services.

Based on the proposed framework, we have implemented a monitoring system of a home network system.

As for the future work, we plan to add features to edit integrated services, which would help developers to create new services, or to restructure the existing service scenarios. Another interesting issue is to monitor the *feature interaction problem* [4], which is known as inconsistent conflicts among integrated services. Since the proposed framework visualizes all the states of the service components, we believe that our monitoring framework is well applicable to detect and resolve the feature interaction problem.

References

[1] Business Process Execution Language for Web Services, Version 1.1. http://www-128.ibm.com /developerworks/library/specification/ws-bpel /

[2] CORBA@BASICS. http://www.omg.org/ gettingstarted/corbafaq.htm

[3] Java Remote Method Invocation (Java RMI). http://java.sun.com/products/jdk/rmi/

[4] Masahide Nakamura, Hiroshi Igaki, Haruaki Tamada, Ken-ichi Matsumoto, "Feature Interacctions in Integrated Services of Networked Home Appliances -An Object Oriented Approach-", *ICFI2005(to appear)*

[5] Masahide Nakamura, Hiroshi Igaki, Haruaki Tamada, Ken-ichi Matsumoto, "Implementing Integrated Services of Networked Home Appliances Using Service Oriented Architecture", *ICSOC'04*, pp269-278, November 2004.

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

[7] W3C Web Service Activity. http://www.w3.org/2002 /ws/

[8] WebSphere Business Integration Monitor. http://www-306.ibm.com/software/integration/wbisf/