RuCAS:
Rule-Based Framework for Managing Context-Aware Services with Distributed Web Services

Hiroki Takatsuka, Graduate School of System Informatics, Kobe University, Kobe, Japan

Sachio Saiki, Graduate School of System Informatics, Kobe University, Kobe, Japan

Shinsuke Matsumoto, Graduate School of System Informatics, Kobe University, Kobe, Japan

Masahide Namamura, Graduate School of System Informatics, Kobe University, Kobe, Japan

ABSTRACT

Machine-to-Machine (M2M) systems and cloud services provide various kinds of data via distributed Web services. A context-aware service recognizes real-world contexts from such data and behaves autonomously. However, it has been challenging to manage contexts and services defined on the heterogeneous and distributed Web services. In this paper, the authors propose a framework, called RuCAS, which systematically creates and manages context-aware service using various Web services. RuCAS describes every context-aware service by an ECA (Event-Condition-Action) rule. For this, an event is a context triggering the service, a condition is a set of contexts to be satisfied for execution, and the action is a set of Web services to be executed by the service. Thus, every context-aware service is managed in a uniform manner. Since RuCAS is published as a Web service, created contexts and services are reusable. As a case study, RuCAS is applied to a real home network system.

Keywords: *Context-Awareness, Event-Condition-Action Rule, Home Network System, Sensor Services, Web Services*

INTRODUCTION

The recent spread of cloud computing and Machine-to-Machine (M2M) technologies allows us to acquire various kinds of data from heterogeneous and distributed systems (Velte, Velte and Elsenpeter, 2010; Wu, Talwar, Johnsson Himayat and Johnson, 2011). The cloud computing provides computational resource and data as networked services, whereas the M2M enables devices to communicate with each other without human intervention. Typical data include temperature, power consumption, weather, system state, operation of a device. Data from the cloud or M2M systems can be obtained usually through Web services or Web-

DOI: 10.4018/IJSI.2015070105

API. Variety of data achieves a *context-aware service* (Cohen et al. 2004), which recognizes a real-world context and behaves autonomously for the context. The context-aware services implement smarter services, which are sensible for the environment and human activities.

Traditionally, the context-aware services had been studied in the field of ubiquitous computing (Randell and Muller, 2000; Gellersen, Schmidt and Beigl, 2002). Many studies were reported on context acquisition, context reasoning and utilization, using ubiquitous sensors deployed on local smart space. Now, the context-aware services must evolve so that the services can deal with global and distributed contexts obtained from Web services of heterogeneous systems (e.g. information services, sensor services, networked appliances, etc.). However, there are few studies adopting distributed Web services for creating context-aware services. In our previous work, we proposed a *sensor service framework* (Nakamura, Matsuo, Matsumoto, Sakamoto and Igaki, 2011), which invokes Web services based on contexts with physical sensors. However, the focus was limited on the sensors only.

Using Web services for inputs and outputs can significantly improve the functionality and flexibility of the context-aware services. However, a major challenge lies in managing complex relations among distributed data sources, defined contexts, and actions caused by the contexts. Unless managed systematically, the service provision would be quite difficult. Therefore, it is essential to have a unified framework for managing advanced context-aware services based on the heterogeneous and distributed Web services.

In this paper, we present a framework called *RuCAS* (*Rule-based management framework for Context-Aware Services*), which systematically creates and manages context-aware service using various Web services. The framework consists of five layers: Web service layer, adapter layer, context layer, action layer and ECA rule layer. The existing Web services for data acquisition are managed in the Web service layer. The data acquisition from heterogeneous Web services is adapted to the standard API in the adapter layer. In the context layer, every context is defined based on the data obtained via the adapter. Every Web service that is triggered by a context is managed in the action layer.

Using these elements, RuCAS defines every context-aware service as an *event-condition-action (ECA) rule*. For this, the event defines a context that triggers a service. The condition refers to a guard condition to execute the service. The action defines Web services executed by the service. Thus, every context-aware service is simply defined and created as a uniformed rule.

To see the feasibility of the proposed method, we conduct a case study that applies the RuCAS framework to creating the context-aware services in a practical home network system. In the case study, it is shown that a smart air-conditioning service with environmental sensors can be easily created by a sequence of RuCAS API.

PRELIMINARIES

Context-Aware Service

A *context* refers to a situational information (e.g. human activity, environment, etc.) derived from information of sensors and systems. A *context-aware service* is a service that automatically detects change of a context and performs appropriate actions corresponding to the context change. For instance, a context "Hot" can be derived from information that "the value of a temperature sensor in a room is higher than 28 degrees". A context-aware service "Automatic Air-conditioning" starts air-conditioning when the context "Hot" holds.

Traditionally, the context-aware services had been studied extensively in the ubiquitous computing area. The conventional studies include a method that uses sensor information to reason contexts in a smart space, and a method that uses smart phone sensors to reason human behavior (Yamamoto, Kouyama, Yasumoto and Ito, 2011; Chon and Cha, 2011).

Obtaining Data from Web Services

The advancement of ICT (Information and Communication Technology) and the Internet enables to obtain various information through the Web. Especially, M2M and Web services play an important role (Wu et al., 2011; Alonso, Casati, Kuno and Machiraju, 2004). The M2M allows various devices to communicate with each other without human intervention. A background of M2M progress is an acceleration of communication and evolution of sensor technology. Used with the cloud computing and big-data processing, M2M is promising to gather real-world contexts that provide values.

Web service is a technology that provides a feature of a system as a service on the Web. Web service can be accessed by a Web standard protocol. The protocol is usually SOAP or REST over HTTP to exchange XML data between a service and a client. The XML-based communication over the Web standard allows developers to integrate distributed and heterogeneous systems. Thus, modern systems often publish own API as a Web service (called Web-API) so that external applications can retrieve information from the system.

In this paper, we focus on modern devices and systems whose internal information can be retrieved via Web services. Our interest is how to create and manage context-aware services using such distributed and heterogeneous Web services.

Home Network System (HNS)

Home Network System (HNS) is a system that provides value added services by connecting household appliances and equipment with the home network (Nakamura, Tanaka, Igaki, Tamada and Matsumoto, 2008; Li and Zhang, 2004). In the HNS, appliances (e.g. TVs, lights, air-conditioners, curtains, fans, etc.) and sensors (e.g. temperature, humidity, illuminance, etc.) are integrated to implement various services and applications.

In our laboratory, we have been developing an actual HNS environment, called *CS27-HNS* (Nakamura et al., 2008). CS27-HNS extensively exploits the concept of *Service Oriented Architecture (SOA)* in order to integrate heterogeneous devices and sensors. We encapsulated vendor-specific operations and communication protocols within Web services. Every device can be operated by Web-API by SOAP or REST protocol. For instance, to change a channel of a TV to 6, a client just accesses a URL http://cs27-hns/TVService/setChannel?channel=6.

Previous Work: Sensor Service Framework

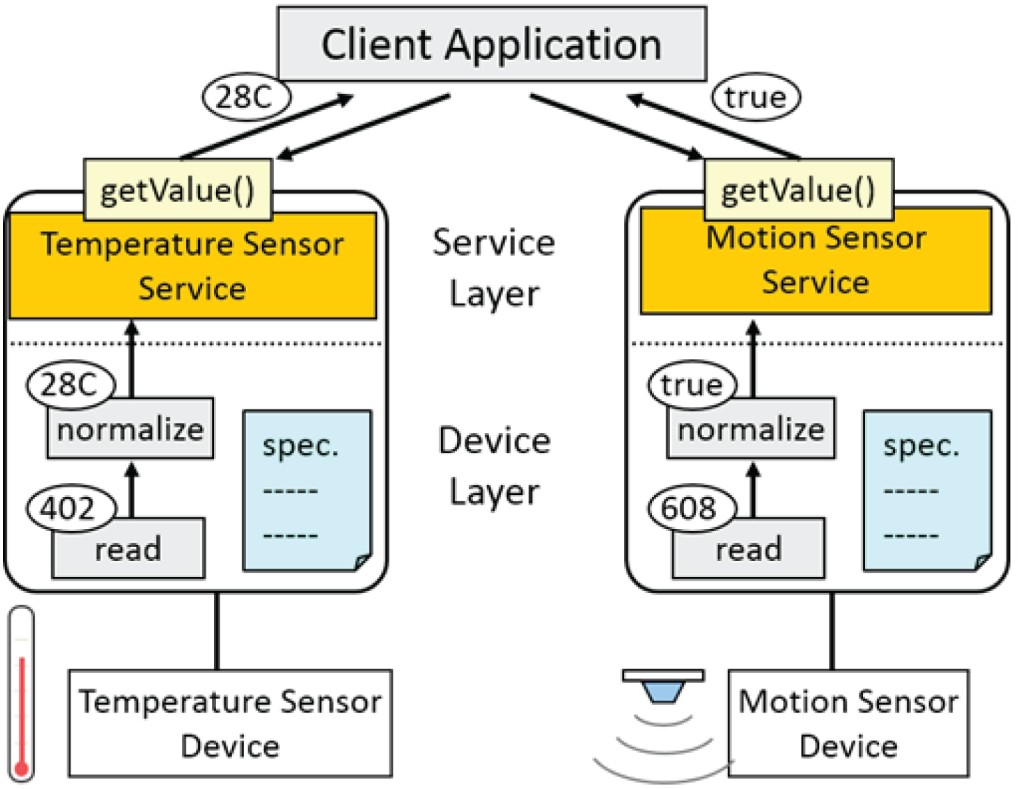
We have previously considered Web services to implement context-aware services in CS27-HNS. *Sensor Service Framework* (SSF) is an application framework that easily deploys environmental sensors (e.g. temperature sensor, illuminance sensor, etc.) as Web services (Nakamura et al., 2011). In SSF, every sensor service has a property representing a standard sensor measure. For instance, a temperature sensor service has temperature property in a degree Celsius. A client can obtain the value of a property by getValue() method, as shown in Figure 1.

Moreover, every sensor service observes the value of the property, and reasons a context based on a registered expression (contextual condition). The registration of the expression is conducted by register() method. For instance, suppose that a client registers a contextual condition Hot: temperature ≥ 28. The registered condition can be bound with an arbitrary Web services by subscribe() method. The sensor service invokes the Web service method when the contextual condition is satisfied. Figure 2 shows a scenario where a client registers and subscribes a context Hot.

Sensor Mashup Platform (SMuP) constructs advanced sensor services by integrating multiple sensor services. *Sensor Service Binder* provides the easy creation of context-aware services with SSF for end users (Nakamura, Matsuo and Matsumoto, 2012).

The above previous methods extensively focused on implementing sensor as a service.

Figure 1. Obtaining sensor values by standard interface of SSF



Thus, using the existing Web services for context-aware services was beyond their scope.

Problem of Creating Context-Aware Services

In the previous methods of context-aware services, every context is tightly coupled with its data source and actions to be invoked, which lacks flexibility and reusability. In many cases, all operations of obtaining data from sensors, evaluating defined contexts and invoking actions are performed within a proprietary program. Hence, it is impossible to reuse a context for another service, or to replace an action with another. In SSF, a context Hot is managed within a temperature sensor service. However, it is not obvious to all clients where the context exists and what happens when Hot becomes true.

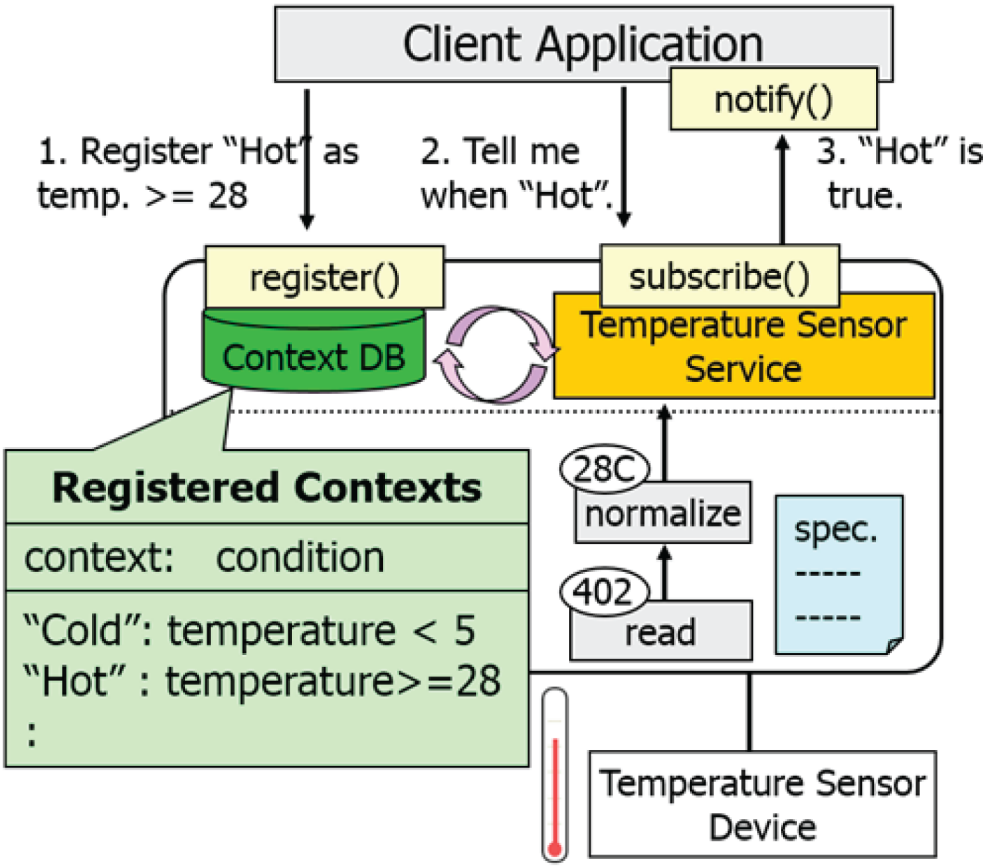
As mentioned before, we aim to implement context-aware services using heterogeneous and distributed Web services, not limited to the conventional sensors. We need to find a way to systematically manage individual Web service, contexts, and context-aware services, in a loose-coupling manner.

RUCAS: FRAMEWORK FOR MANAGING CONTEXT-AWARE SERVICES WITH WEB SERVICES

Overview

To cope with the challenge, we propose a framework, *RuCAS (Rule-based management framework for Context-Aware Services)*, which

Figure 2. Implementing context-aware service with SSF



creates and manages context-aware services based on various Web services.

RuCAS supports client applications to acquire information from heterogeneous and distributed Web services, and to define and manage contexts based on the information. In addition, RuCAS defines every context-aware service as an *ECA (Event-Condition-Action) rule*, where the *event* is a satisfaction of a context triggering the service, the *condition* is a guard condition enabling the service, and the *action* is Web services to be executed.

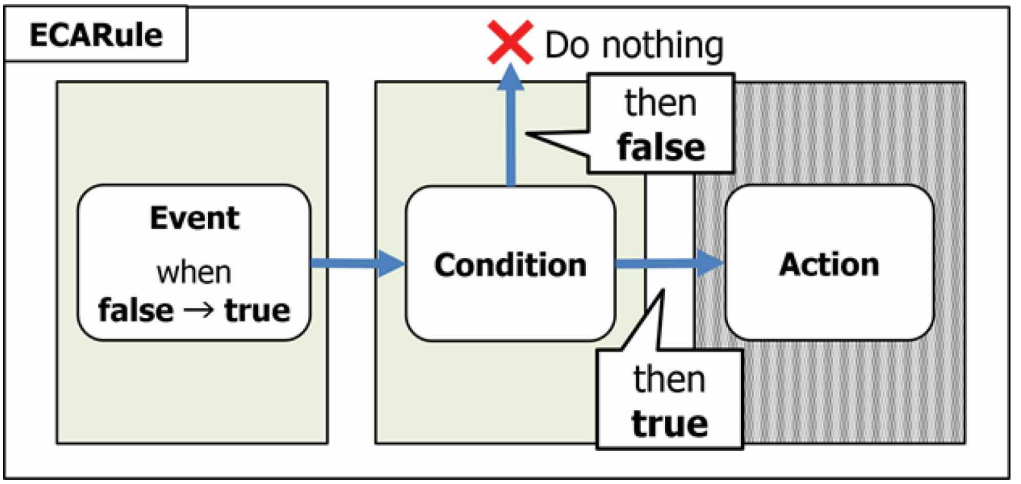
By using RuCAS, every context-aware service can be uniformly described by a rule, which combines defined contexts and actions. Thus, RuCAS achieves loose coupling of Web services as a source of contexts, contexts defined with the data sources and context-aware services with actions. This enables flexible creation and management of context-aware services.

Event-Condition-Action (ECA) Rule

The *ECA rule* is an important design through of RuCAS, which defines every context-aware service as a set of [Event, Condition, Action]. In general, a context-aware service can be described by a rule that "when a context becomes true, do something". Intuitively, the part "when a context becomes true" corresponds to the *event*, whereas "do something" corresponds to the *action* in RuCAS.

However, the above rule lacks flexibility, since the action always fires when the context

Figure 3. Semantics of ECA Rule



becomes true. Therefore, we extend the rule a bit such that "when a context becomes true, if a condition is satisfied, do something". The part "if a condition is satisfied" corresponds to the *condition* in RuCAS. More specifically, in this paper, we define a context, an event, a condition and an action as follows.

1. A *context* is a situational information defined by a logical expression over data obtained from a Web service. Depending on the value of the data, every context is evaluated to true or false. A context can be also defined by a composition of the existing contexts.
2. An *event* is a context triggering the execution of a context-aware service.
3. A *condition* is a guard condition enabling the execution of a context-aware service. A condition is defined by one or more contexts.
4. An *action* is operations executed by a context-aware service. An action is defined by one or more Web services.

Then, an ECA rule is defined as follows:

- **ECA Rule:** Let c1, c2, ... be contexts, and let a1, a2, ... be invocations of Web services.

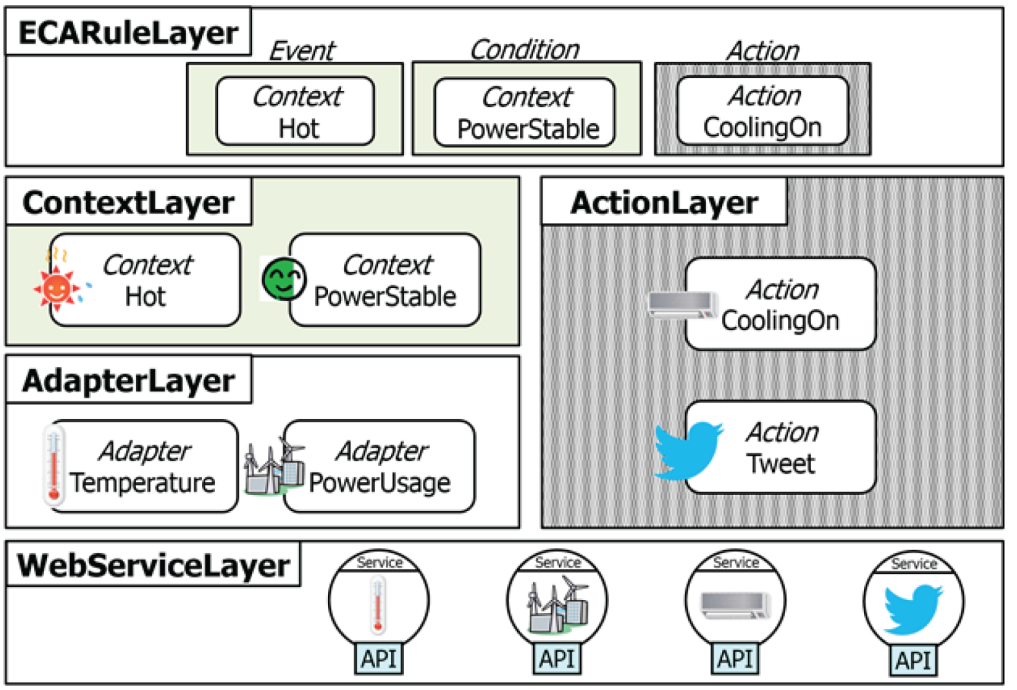
An ECA rule r is defined by r = [E: ci, C: {$cj1, cj2, ..., cjm$}, A: {$ak1, ak2, ..., akn$}], where E is an event, C is a condition, A is an action. For r, we say "event E occurs" if the value of context ci moves from false to true. When E occurs, if all contexts $cj1, cj2, ..., cjm$ are satisfied, we say "r is executed". When r is executed, all Web services $ak1, ak2, ..., akn$ are invoked.

Figure 3 shows semantics of the ECA rule. An event is defined by a single context, and occurs when the context moves from false to true. A condition defines a guard evaluated when the event occurs, If the condition is not satisfied, no action is performed. If satisfied, the action is executed to invoke Web services. For instance, a context-aware service "when it is hot, if a user is present in a room, turn on an air-conditioner" can be described by an ECA rule: [E: Hot, C: {$PresentUser$}, A: {$AirCon.on$}].

System Requirement of RuCAS

We determine the following requirements R1 to R4 to implement RuCAS.

Figure 4. Architecture of RuCAS



R1: The framework should be able to create contexts using information from the existing Web services.

R2: The framework should be able to create actions using an invocation of the existing Web services.

R3: The framework should be able to create context-aware services as ECA rules using the contexts and actions.

R4: The framework should be able to use, update and delete the created method.

Architecture of RuCAS

Figure 4 shows the architecture of RuCAS. In order to efficiently build ECA rules from existing elements, RuCAS consists of five layers: Web service layer, adapter layer, context layer, action layer and ECA rule layer. Each layer creates and manages elements using features of an underlying layer. In the ECA rule layer at the top, RuCAS defines every context-aware service as an ECA rule, by combining existing elements created in underlying layers. Features of each layer are described below.

Web Service Layer

The Web service layer manages the existing Web services used as input or output of context-aware services. The input Web service is a Web service that can return a certain value (numeric, Boolean, string, etc.) for defining a context. Typical examples include the conventional sensor services, the status of a device, dynamic Web information (weather, stock price, exchange rate, etc.), SNS, clock, system logs. The output Web service is a Web service that can yield an action. Examples include an operation of home network system (switch on/off, voice announce, etc.) and a request to an information system or service (send an email, post a comment to SNS, etc.).

Adapter Layer

To obtain data from a Web service, a client needs to invoke Web-API and extract the necessary data by parsing the return value. However, Web-API and the return value vary from a Web service to another. Therefore, the adapter layer creates an adapter that normalizes the heterogeneous interface. Specifically, every Web-API used to obtain data is adapted to uniform API getValue().

For example, we can create an adapter TempAdapter, by using a temperature sensor Web service, say http://cs27-hns/TemperatureSensorService/getTemperature. Within RuCAS, TempAdapter.getValue() returns a temperature by internally invoking the Web service.

Context Layer

The context layer manages all contexts defined by data from Web services via the adapter layer. In this layer, every context is defined by *context ID* and *context expression*. The context ID is a label to identify every context. The context expression is a logical formula, in form of Adapter.value comp_op const, where comp_op is a comparative operator and const is a constant value. For example, to define Hot context to be "the temperature is equal to or more than 28 degrees", RuCAS describes it by [Hot: TempAdapter.value >= 28]. Similarly, to define Humid to be "the humidity is equal to or more than 70 percent", RuCAS describes it by [Humid: HumidAdapter.value >= 70]. Each context can be associated with a *refresh interval*, by which RuCAS periodically evaluates the context expression. For example, when the refresh interval of Hot is one minutes, RuCAS obtains a new value from TempAdapter and evaluates the truth value of Hot every one minutes.

RuCAS can define two types of contexts: *atomic* and *compound*. The atomic context is a context directory defined by a single Web service. The compound context is a context defined by the existing contexts combined with logical operators (!: NOT, &&: AND, ||: OR). For example, a compound context Muggy can be defined by combining Hot and Humid such that [Muggy: Hot && Humid].

Action Layer

The action layer manages all actions used in ECA rules. Every action wraps an output Web service of a context-aware service, and is defined by an endpoint, a method name, and parameters of the Web service. Each action is associated with *action ID*, by which RuCAS invoke the Web service as an action. For example, we can create an action CoolingOn, by using an air-conditioner Web service, say http://cs27-hns/AirConService/on?mode=cooling.

When RuCAS invokes CoolingOn, the Web service is executed to turn on an air-conditioner with a cooling mode.

ECA Rule Layer

The ECA rule layer defines a context-aware service as an ECA rule by using contexts in the context layers and actions in the action layer. An ECA rule can be created as follows:

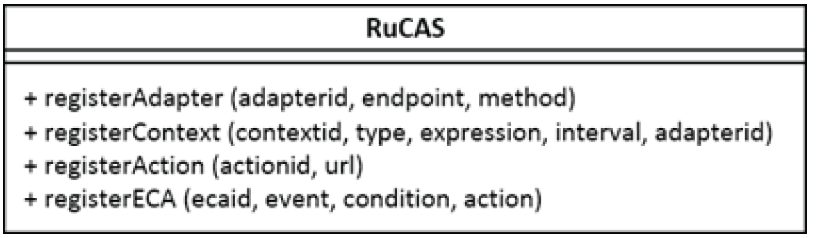
1. Define an event by choosing a single context from the context layer.
2. Define a condition by choosing one or more contexts from the context layer.
3. Define an action by choosing one or more actions from the action layer.

The created ECA rule is evaluated and executed by RuCAS, based on the semantics defined before.

API of RuCAS

Figure 5 shows the typical API of RuCAS, registering a new element to a layer. registerAdapter() creates a new adapter with adapter ID, endpoint and method of a Web service. registerContext() creates a new context with context ID, type to specify atomic (A) or compound (C), context expression, refresh interval (in msec), and adapter ID used to obtain data. registerAction() creates a new action with action ID and URL of a Web service. registerECA() creates a new

Figure 5. Method of RuCAS



ECA rule with ECA rule ID, event given by a context ID, condition given by a set of context IDs, action given by a set of action IDs.

Based on the concept of SOA, the above API is deployed as a Web service so that various clients can easily create and manage their own context-aware services. For example, to create TempAdapter mentioned before, a client just accesses the following URL: http://RuCAS/registerAdapter?adapterid=TempAdapter&endpoint=http://cs27-hns/TemperatureSensorService&method=getTemperature

Creating Context-Aware Service with RuCAS

Using RuCAS, we can easily create a context-aware service by the following four steps:

Step 1 (Creating Adapters): Define adapters by registerAdapter() with interesting Web services.

Step 2 (Creating Contexts): Using the adapters, define necessary contexts by registerContext().

Step 3 (Creating Actions): Define actions by registerAction() with Web services to be executed.

Step 4 (Creating ECA Rule): Define an ECA rule by registerECA() with the created contexts and actions.

CASE STUDY

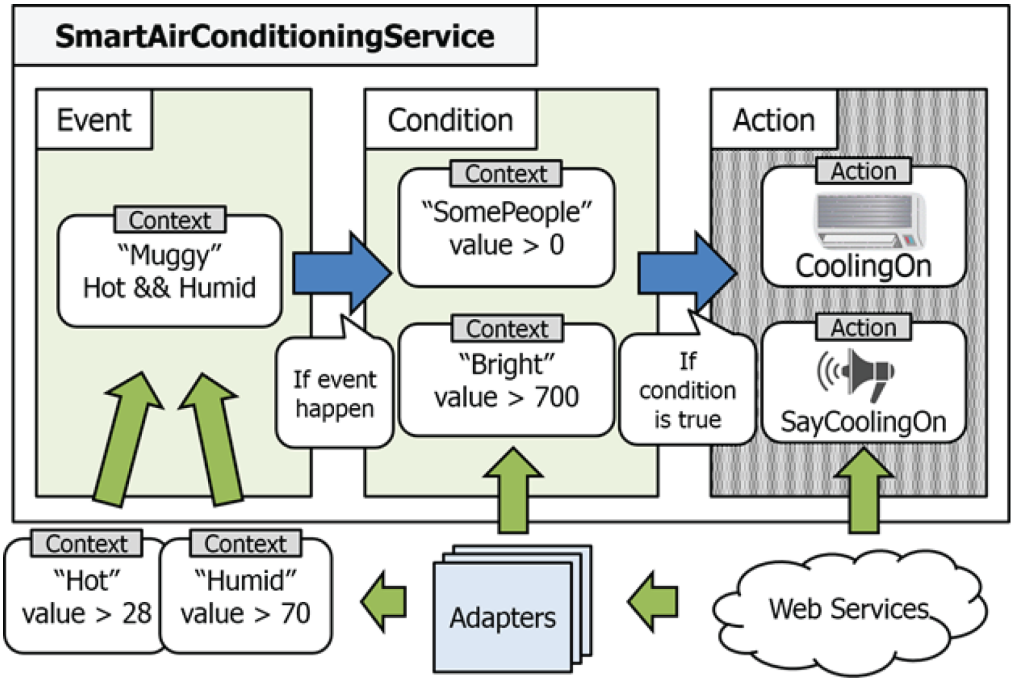
To demonstrate the proposed framework, we create a *smart air-conditioning service* using RuCAS. The definition of the service is as follows: "when a room becomes muggy, if some people are present in the room and the room is bright enough, turn on an air-conditioner and announce the service". The service is supposed to be implemented in a real environment of CS27-HNS. Using Web services of CS27-HNS, we create contexts and actions within RuCAS.

Figure 6 shows the outline of service creation. We create the service as an ECA rule such that [E: Muggy, C: {SomePeople, Bright}, A: {CoolingOn, SayCoolingOn}]. In the rule, Muggy is a compound context defined by Hot and Humid, where Hot is an atomic context that "temperature is greater than 28 degrees", and Humid is an atomic context that "humidity is greater than 70 percent". In the condition, SomePeople is an atomic context that "the number of people is greater than 0", and Bright is that "the illuminance is greater than 700 lux". Finally, CoolingOn is an action that "turn on an air-conditioner", and SayCoolingOn is an action that speaks "Starting air-conditioning." to announce the users.

As seen in the previous section, we create the service by the following four steps.

Step 1 (Creating Adapters): To create the context, we use sensor Web services of CS27-HNS, including a temperature sensor, a humidity sensor and an illuminance sensor. We also use InOutUserManageService to get the number of people within the room. Using registerAdapter() of RuCAS, we create four adapters Temperature, Humidity, Illuminance, PeopleCounter by specifying parameters in Table 1.

Figure 6. Outline of creating smart air-conditioning service



Step 2 (Creating Contexts): We first create four atomic contexts Hot, Humid, Bright, SomePeople using the four adapters Temperature, Humidity, Illuminance, PeopleCounter, respectively. Then, a compound context Muggy is created with Hot and Humid. These contexts are created by registerContext() of RuCAS by specifying parameters in Table 2.

Step 3 (Creating Actions): We use the AirCondtioner Web service and SpeechToText Web service to define CoolingOn and SayCoolingOn. These actions created by registerAction() of RucAS by specifying parameters in Table 3.

Step 4 (Creating ECA Rule): We create the ECA rule [E: Muggy, C: {SomePeople, Bright}, A: {CoolingOn, SayCoolingOn}] using the created contexts and actions. The ECA rule is created by registerECA() of RuCAS by specifying parameters in Table 4.

We implemented a prototype of RuCAS, and confirmed that the smart air-conditioning service works fine in CS27-HNS. Due to limited space, the details of design and implementa-

Table 1. Parameters of registerAdapter()

adapterid	endpoint	method	description
Temperature	http://cs27-hns/TemperatureSensorService	getValue	Get temperature.
Humidity	http://cs27-hns/HumiditySensorService	getValue	Get humidity.
Illuminance	http://cs27-hns/IlluminanceSensorService	getValue	Get illuminance.
PeopleCounter	http://cs27-hns/InOutUserManageService	getHeads	Get a number of person in the room.

Table 2. Parameters of registerContext()

contextid	type	expression	interval	adapterid	description
Hot	A	value>28	5,000	Temperature	Temperature is over 28 degrees.
Humid	A	value>70	5,000	Humidity	Humidity is over 70 percent.
SomePeople	A	value>0	10,000	PeopleCounter	There are people in the room.
Bright	A	value>700	5,000	Illuminance	Illuminance is over 700 lux.
Muggy	C	Hot&&Humid	5,000	--	Hot and Humid are true.

Table 3. Parameters of registerAction()

actionid	url	description
CoolingOn	http://cs27-hns/AirConditionerService/on?mode=cooling	Turn on the air conditioner.
SayCoolingOn	http://cs27-hns/TextToSpeechService?text=starting air conditioning	The system says "Starting air conditioning.".

tion of RuCAS will be discussed in our future publications.

CONCLUSION

In this paper, we have proposed a rule-based framework RuCAS for creating and managing context-aware services with distributed and heterogeneous Web services. Using RuCAS, every context-aware service is uniformly defined by an ECA rule. Every ECA rule is assembled by a loose coupling of Web services, contexts and actions, coordinated by five layers of RuCAS. A case study showed that a practical context-aware service can be implemented easily by invoking RuCAS API.

Our future work includes implementation of a service platform of RuCAS and user support tools (e.g. GUI and manuals). We also plan to conduct an experimental evaluation of service creation. The service interaction problem is also an important issue to guarantee the consistency among multiple user-made services (Wilson, Kolberg and Magill, 2008).

ACKNOWLEDGMENT

This research was partially supported by the Japan Ministry of Education, Science, Sports, and Culture [Grant-in-Aid for Scientific Research (C) (No.24500079, No.24500258), Scientific Research (B) (No.26280115), Young Scientists (B) (No.26730155)] and Kawanishi Memorial ShinMaywa Education Foundation.

Table 4. Parameters of registerECA()

ecaid	event	condition	action	description
SmartAirConditioningService	Muggy	[SomePeople, Bright]	[SayCoolingOn, CoolingOn]	Automatic air-conditioning service.

REFERENCES

Alonso, G., Casati, F., Kuno, H., & Machiraju, V. (2004). Web Services. Berlin, German: Springer Berlin Heidelberg. doi:10.1007/978-3-662-10876-5

Chon, J., & Cha, H. (2011). Lifemap: A smartphone-based context provider for location-based services. *IEEE Pervasive Computing / IEEE Computer Society [and] IEEE Communications Society*, *10*(2), 58–67. doi:10.1109/MPRV.2011.13

Cohen, N. H., Black, J., Castro, P., Ebling, M., Leiba, B., Misra, A., & Segmuller, W. (2004). *Building context-aware applications with context weaver*. NY, US: IBM Research Division, TJ Watson Research Center.

Gellersen, H. W., Schmidt, A., & Beigl, M. (2002). Multi-sensor context-awareness in mobile devices and smart artifacts. *Mobile Networks and Applications*, *7*(5), 341–351. doi:10.1023/A:1016587515822

Li, X., & Zhang, W. (2004). The design and implementation of home network system using OSGi compliant middleware. *IEEE Transactions on Consumer Electronics*, *50*(2), 528–534. doi:10.1109/TCE.2004.1309419

Nakamura, M., Igaki, H., Yoshimura, Y., & Ikegami, K. (2009). Considering Online Feature Interaction Detection and Resolution for Integrated Services in Home Network System. *International Conference on Feature Interactions in Telecommunications and Software Systems* (pp. 191-206).

Nakamura, M., Matsuo, S., & Matsumoto, S. (2013). Supporting end-user development of context-aware services in home network system. *Software Engineering, Artificial Intelligence, Networking and Parallel* [Springer Berlin Heidelberg.]. *Distributed Computing*, *2012*, 159–170.

Nakamura, M., Matsuo, S., Matsumoto, S., Sakamoto, H., & Igaki, H. (2011). Application framework for efficient development of sensor as a service for home network system. *IEEE International Conference on Services Computing* (pp. 576-583). IEEE. doi:10.1109/SCC.2011.18

Randell, C., & Muller, H. (2000). Context awareness by analysing accelerometer data. *The Fourth International Symposium on Wearable Computers* (pp. 175-176). IEEE. doi:10.1109/ISWC.2000.888488

Velte, T., Velte, A., & Elsenpeter, R. (2009). *Cloud computing, a practical approach*. New York, USA: McGraw-Hill, Inc.

Wilson, M., Kolberg, M., & Magill, E. (2008). Considering Side Effects in Service Interactions in Home Automation-An Online Approach. [IOS Press.]. *Feature Interactions in Software and Communication Systems*, *IX*, 172–187.

Wu, G., Talwar, S., Johnsson, K., Himayat, N., & Johnson, K. D. (2011). M2M: From mobile to embedded internet. *IEEE Communications Magazine*, *49*(4), 36–43. doi:10.1109/MCOM.2011.5741144

Yamamoto, S., Kouyama, N., Yasumoto, K., & Ito, M. (2011). Maximizing users comfort levels through user preference estimation in public smartspaces. *IEEE International Conference on Pervasive Computing and Communications Workshops 2011* (pp. 572-577). IEEE. doi:10.1109/PERCOMW.2011.5766955