Design and Implementation of Rule-Based Framework for Context-Aware Services with Web Services

Hiroki Takatsuka, Sachio Saiki, Shinsuke Matsumoto, Masahide Nakamura
Graduate School of System Informatics, Kobe University
tktk@ws.cs.kobe-u.ac.jp, sachio@carp.kobe-u.ac.jp, shinsuke@cs.kobe-u.ac.jp,
masa-n@cs.kobe-u.ac.jp

ABSTRACT

Modern cloud services and machine-to-machine (M2M) systems provide various kinds of data via various Web services. Implementing *context-aware services* integrating such global data are promising in various applications. However, it has been challenging to manage heterogeneous contexts and services defined in various Web services. To cope with this, we design a framework, called *RuCAS*, which systematically manages every context-aware service in form of *ECA (Event-Condition-Action) rule*. We also develop *RuCAS platform*, which publishes API of RuCAS as Web service. Using the RuCAS platform, users can define their own contexts with various Web services (e.g., information service, sensor services, networked appliances, etc.). Based on the defined contexts, they can create ECA rules to define custom context-aware services. To support users, We also implement a GUI front-end of RuCAS platform, called *RuCAS.me*. RuCAS.me supports users even if the users are non-expert. A case study in a real home network system demonstrates practical feasibility of RuCAS platform and RuCAS.me. The contribution of this paper is to provide design and implementation details of RuCAS, by which one can fully understand systematic management of context-aware services with Web services.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed applications; C.0 [**General**]: System architectures; C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems

General Terms

Design

Keywords

Web services, context-awareness, event-condition-action rule, home network system, sensor services

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
iiWAS2014, 4-6 December, 2014, Hanoi, Vietnam.
Copyright 2014 ACM 978-1-4503-3001-5/14/12 ...$15.00.

1. INTRODUCTION

The recent spread of cloud computing [17] and Machine-to-Machine (M2M) technologies [19] allows us to acquire various kinds of data from heterogeneous and distributed systems. 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 often obtained through Web services or Web-API. Variety of data achieves *context-aware services* [6]. A context-aware service recognizes a real-world context and performs an appropriate action based on the context. Traditionally, the context-aware services had been studied in the field of ubiquitous/pervasive computing [5] [20]. Many studies were reported on context acquisition, context reasoning and utilization, using ubiquitous sensors deployed on local smart space.

Now in the era of cloud and M2M, the context-aware services must evolve so that the services can deal with *global contexts*, which are defined by integrating data from the various Web services (e.g., information services, sensor services, networked appliances, etc.). However, how to manage various Web services for creating context-aware services is still an open issue. Although there exist relevant studies or services (discussed in Section 7.2), 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 various Web services.

To cope with the challenge, we design a framework called *RuCAS (Rule-based management framework for Context-Aware Services)*. The RuCAS framework pulls data from the existing Web services, and defines custom contexts based on the data. It also defines actions that executes assigned Web services. Using the custom contexts and actions, RuCAS describes *ECA (Event-Condition-Action) 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 managed as a uniformed rule. RuCAS consists of five layers: Web service layer, adapter layer, context layer, action layer and ECA rule layer. The Web service layer refers to the existing Web services as a source of data and actions. 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 as an expression over the data obtained via the adapter. Every Web service that is triggered by a context is managed in the action layer. Finally, the ECA rule layer constructs ECA rules binding the custom contexts and actions. We also develop a service platform, called *RuCAS platform*, which provides API of RuCAS as Web service. Using the RuCAS platform, various client applications can define their own contexts and context-aware services using various Web services.

In addition, to support even non-expert users who are unfamiliar with Web services programming, we also implement a GUI front-end of RuCAS platform, called *RuCAS.me*. Based on an intuitive user interface, a user can easily create and manage adapters, contexts, actions, and ECA rules within the RuCAS platform. The use of RuCAS.me significantly reduces the effort of service creation.

In order to evaluate the practical feasibility, we conduct a case study using the RuCAS platform and RuCAS.me. Within an actual home network system, we implement a sustainable air-conditioning service. It controls an air-conditioner and a fan based on contexts defined over room temperature, humidity and regional energy consumption. It is demonstrated that the service can be implemented efficiently using RuCAS.me.

The original concept of RuCAS was published in [16], and discussion in the context of self-management system was submitted to a workshop paper [15]. Changes were made on this full-conference paper, most significantly the addition of design and implementation details (in Sections 4 and 5), and elaborate discussion (in Section 7). We believe that those changes will help practitioners to fully understand the systematic management of context-aware services with Web services, and to develop similar systems, efficiently.

2. PRELIMINARIES

2.1 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 a change of contexts 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". An example of context-aware service, say "Automatic Air-conditioning", turns on an air-conditioner when the context "Hot" holds.

Traditionally, the context-aware services had been studied extensively in ubiquitous/pervasive computing. Contexts were generally defined using situational sensors in a smart space [20] or hand-held devices (e.g., smartphone) [5]. However, the recent advancement of cloud computing [17] and M2M technologies [19] dramatically extends the scope of context-aware services. The cloud computing enables to share data and resources as services in the Web. The M2M allows various devices to communicate with each other without human intervention. Supported by big-data processing, the combination of the cloud and the M2M is promising to gather real-world contexts from the entire globe. Thus, the context-aware services can evolve to sense global contexts.

To facilitate data sharing and resource integration, modern systems and devices often exhibit Web services (i.e., Web-API) that publish internal data and operations. This paper focuses on such modern systems, and considers how to create and manage context-aware services with such Web services.

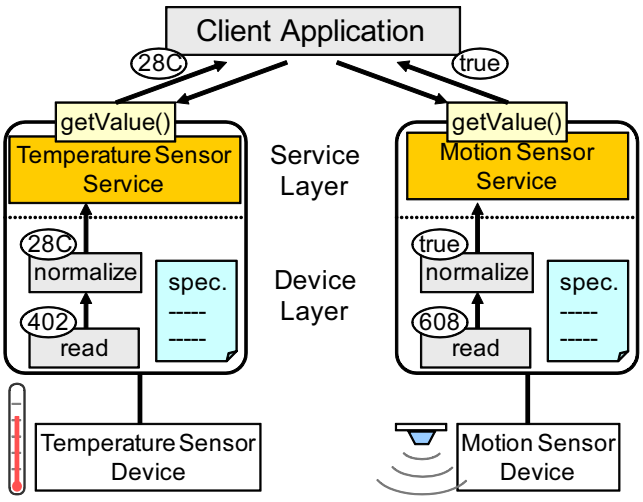

Figure 1: Obtaining sensor values by standard interface of SSF

2.2 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 [7] [11]. 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* [11]. 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://hns/TVService/setChannel?channel=6`.

2.3 Sensor Service Framework [10]

We have previously considered Web services to implement context-aware services in CS27-HNS. *Sensor Service Framework (SSF)* [10] is an application framework that easily deploys environmental sensors (e.g., temperature sensor, illuminance sensor, etc.) as Web services. 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 an expression (context expression). The registration of the expression is conducted by `register()` method. For instance, suppose that a client registers a context `Hot: temperature` $\geq$ `28`. The registered context can be bound with an arbitrary Web services by `subscribe()` method. The sensor service invokes

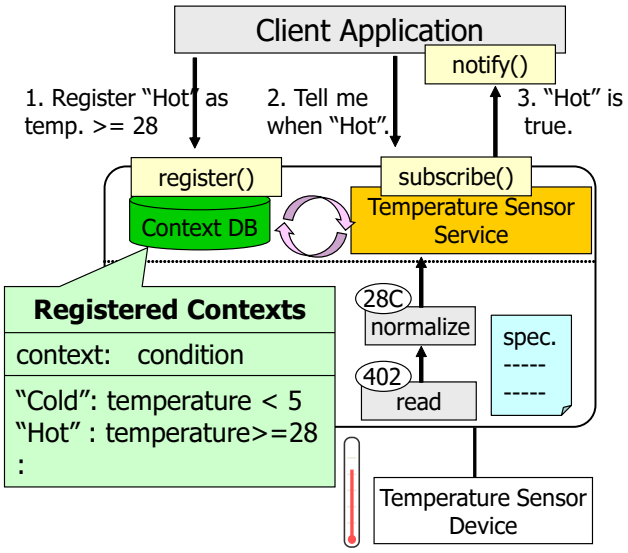

Figure 2: Implementing context-aware service with SSF

the Web service method when the context is satisfied. Figure 2 shows a scenario where a client registers and subscribes a context `Hot`.

Sensor Mashup Platform (SMuP) [10] constructs advanced sensor services by integrating multiple sensor services. *Sensor Service Binder* [9] provides the easy creation of context-aware services with SSF for end users.

The above previous methods extensively focused on implementing sensor as a service. Thus, using the existing Web services for context-aware services was beyond their scope.

2.4 Challenge in Context-Aware Services with Web 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. For instance, in SSF, a context `Hot` is managed within a temperature sensor service. However, it is not obvious for all clients where the context exists and what happens when `Hot` becomes true.

As mentioned in Section 2.1, we aim to implement context-aware services using 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.

3. RUCAS: RULE-BASED FRAMEWORK FOR CONTEXT-AWARE SERVICES

To cope with the challenge, we propose a concept of *RuCAS (Rule-based management framework for Context-Aware Services)*, which creates and manages context-aware services based on various Web services.

RuCAS aims to help client applications to acquire information from heterogeneous and various Web services, and

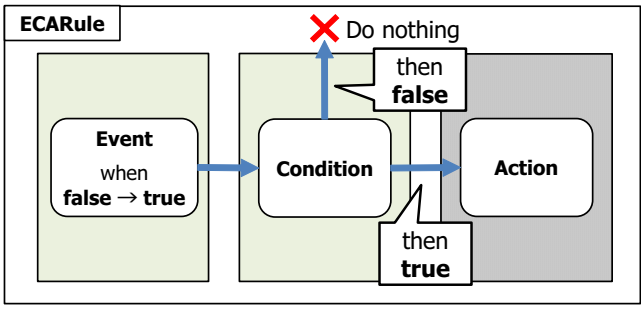

Figure 3: Semantics of ECA rule

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.

The *ECA rule* is an important design principle 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 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.

We defined a context, an event, a condition and an action as follows.

- 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.
- An *event* is a context triggering the execution of a context-aware service.
- A *condition* is a guard condition enabling the execution of a context-aware service. A condition is defined by one or more contexts.
- An *action* is operations executed by a context-aware service. An action is defined by one or more Web services.

Then, an ECA rule was defined as follows:

- *ECA Rule:* Let $c_1, c_2, ...$ be contexts, and let $a_1, a_2, ...$ be invocations of Web services. An ECA rule r is defined by $r = [E : c_i,\ C : \{c_{j_1}, c_{j_2}, ..., c_{j_m}\},\ A : \{a_{k_1}, a_{k_2}, ..., a_{k_n}\}]$, 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 c_i moves from false to true. When E occurs, if all contexts $c_{j_1}, c_{j_2}, ..., c_{j_m}$ are satisfied, we say "r is executed". When r is executed, all Web services $a_{k_1}, a_{k_2}, ..., a_{k_n}$ 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: $[E : Hot, C : \{PresentUser\}, A : \{AirCon.on\}]$. Thus, the ECA rules give an intuitive but systematic foundation to define context-aware services.

4. DESIGN AND IMPLEMENTATION OF RUCAS AS SERVICE PLATFORM

We then design and implement RuCAS as a *service platform*, with which various clients can manage their own contexts and services via a network. The implementation is deployed as a cloud service, which we call *RuCAS platform*.

4.1 System Architecture

Figure 4 shows the architecture of the RuCAS platform. In order to efficiently build ECA rules from existing elements, the platform 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.

4.1.1 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 (e.g., numeric, Boolean, string, etc.) for defining a context. Typical examples include the conventional sensor services, the status of a device, dynamic Web information (e.g., 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 (e.g., switch on/off, voice announce, etc.) and a request to an information system or service (e.g., send an email, post a comment to SNS, etc.).

4.1.2 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://hns/TemperatureSensorService/getTemperature`. Within RuCAS, `TempAdapter.getValue()` returns a temperature by internally invoking the Web service.

4.1.3 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

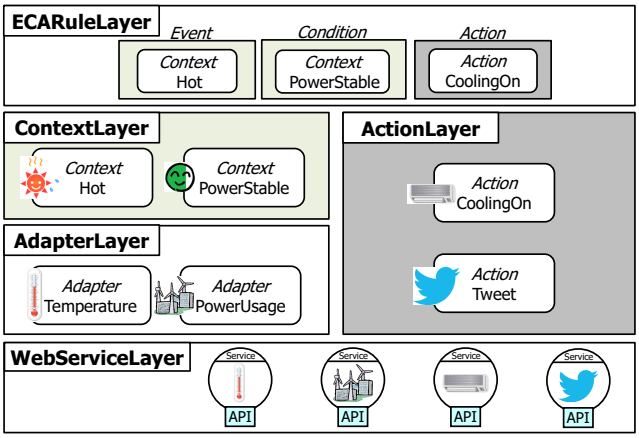

Figure 4: Architecture of RuCAS platform

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 minute, RuCAS obtains a new value from `TempAdapter` and evaluates the truth value of `Hot` every minute.

RuCAS can define two types of contexts: *atomic* and *compound*. The atomic context is a context directly 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]`.

4.1.4 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://hns/AirConService/on?mode=cooling`. When RuCAS invokes `CoolingOn`, the Web service is executed to turn on an air-conditioner with a cooling mode.

4.1.5 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 in Section 3.

4.2 Detailed Design

Based on the architecture, we conduct object-oriented design of the RuCAS platform. Figure 5 shows a class diagram. This section describes the detail of each class.

4.2.1 Adapter

`Adapter` class implements an adapter of RuCAS. As mentioned in Section 4.1.2, it has `getValue()` method that internally obtains a return value from an external Web service. Properties `endpoint` and `method` are used to invoke the Web service. If the return value is structured data with multiple attributes, `property` specifies an attribute to be extracted from the structure. The followings summarize primary properties and methods.

`adapterid:` identifier of the adapter.

`endpoint:` URL endpoint of a Web service.

`method:` Web-API name of the Web service.

`property:` an attribute to be extracted from a structured return value of the web service.

`getValue():` returns a value of the property obtained from the Web service.

4.2.2 Context

`Context` class implements a context of RuCAS. As mentioned in Section 4.1.3, there are two types of contexts: `atomic` and `compound`. `Context` is an abstract class and is implemented by either `AtomicContext` or `CompoundContext`. Method `run()` is executed every `interval` msec to refresh the context. The `run()` method internally executes `resolve()` to obtain the latest values of variables and assign the values to the context expression (`expression`). Then, it executes `eval()` to evaluate the context expression. The resultant value (true or false) is stored in `present`, until the next execution of `run()`. When the value of `present` changes from false to true, an event defined by this context occurs. The event is notified to all the relevant ECA rules by `notifyECA()`. The followings summarize primary properties and methods.

`contextid:` identifier of the context.

`type:` type of context: atomic (A) or compound (C).

`expression:` a context expression of this context.

`interval:` refresh interval (in msec).

`ecaarray:` pointers to ECA rules, whose events are defined by this context.

`present:` present value of the context (true or false).

`eval():` evaluate the `expression`.

`run():` refresh variables the context value.

4.2.3 AtomicContext

`AtomicContext` class implements the atomic context of RuCAS. It has an `adapter` to obtain data from an external Web service. In `resolve()` method, the context invokes `getValue()` of the `adapter` and assigns the obtained value into a variable in the context expression (a part of the string `value`, see Section 4.1.3). The followings summarize primary properties and methods.

`adapter:` an adapter to obtain data from Web service.

`resolve():` assign a value from the adapter to the context expression.

4.2.4 CompoundContext

`CompoundContext` class implements the compound context of RuCAS. It contains multiple child contexts, which are stored in `children` property. In `resolve()` method, each child context of `children` recursively executes `resolve()` to obtain truth value of the child context. The followings summarize primary properties and methods.

`children:` array of child contexts contained in the context expression.

`resolve():` assign values obtained from the child contexts in variables in the context expression.

4.2.5 Action

`Action` class implements the action of RuCAS. As mentioned in Section 4.1.4, it has a `url` that contains an endpoint, a method and an argument of a Web service to be invoked. The followings summarize primary properties and methods.

`actionid:` identifier of the action.

`url:` URL endpoint of a Web service.

`invokeUrl():` invoke the Web service.

4.2.6 ECA

`ECA` class implements the ECA rule of RuCAS. Every ECA rule is executed based on the semantics defines in Section 3. When a context assigned as an event of an ECA rule moves from false to true, the context invokes `notifyECA()` of the rule to tell that the event occurs. In `notifyECA()` method, the ECA rule evaluates `condition` based on the current value of the contexts. If `condition` is true, the rule executes `invokeUrl()` to invoke Web services specified in `action`. The followings summarize primary properties and methods.

`ecaid:` identifier of the ECA rule.

`event:` event of the ECA rule (defined by a context).

`condition:` condition of the ECA rule (defined by multiple contexts).

`action:` a set of action of the ECA rule.

`notifyECA():` triggers the ECA rule when an event occurs.

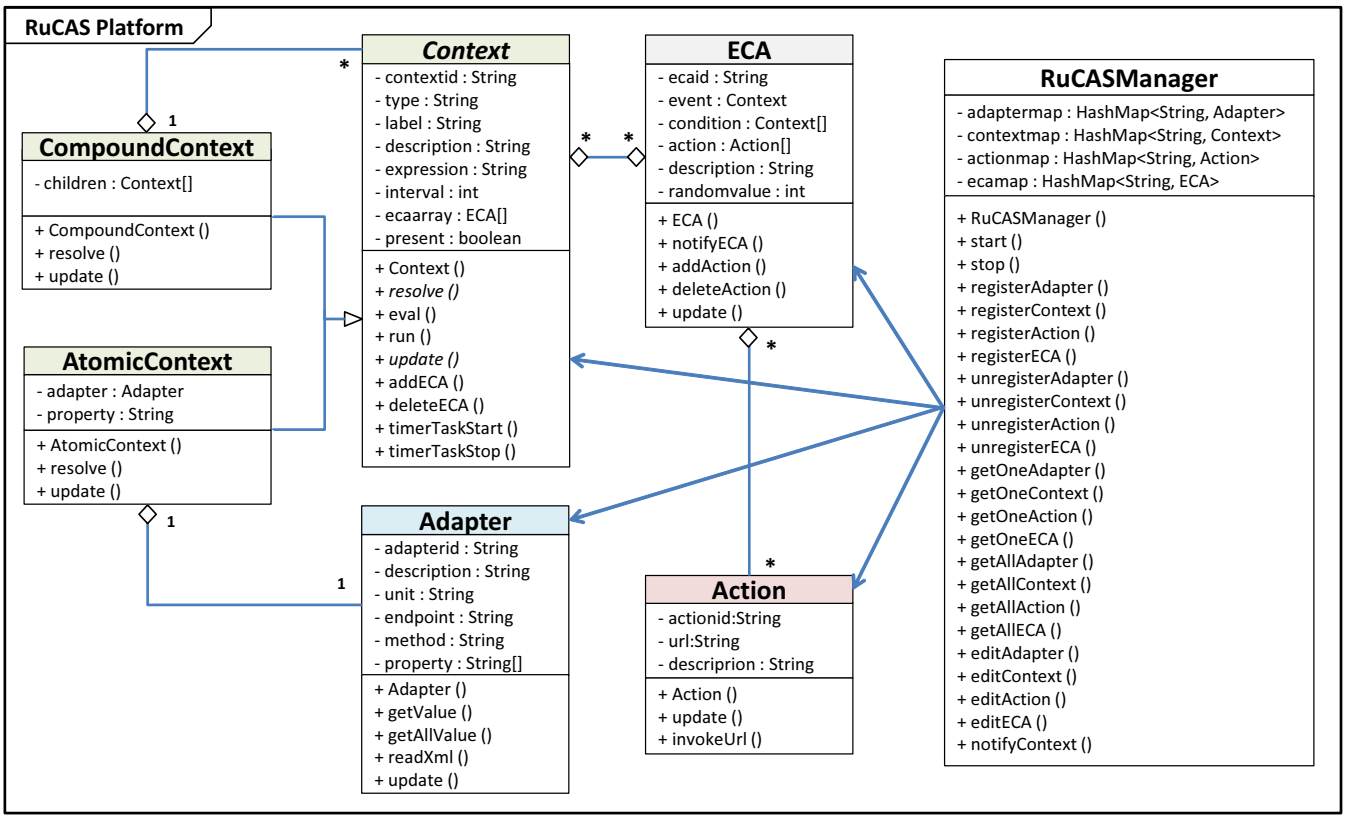

Figure 5: Class diagram of RuCAS platform

4.2.7 *RuCASManager*

`RuCASManager` class works as a façade of all the above classes. It provides a service interface of the RuCAS platform. The method includes CRUD operations (register, get, edit and unregister) for `Adapter`, `Context`, `Action` and `ECA`. It can also start and stop the evaluation process of contexts. The method `notifyContext()` updates a designed context to be true. Using the method, the context can be updated based on publish/subscribed message pattern, instead of the periodical polling.

In the following, we explain method that register new objects within the RuCAS platform.

`registerAdapter(adapterid,endpoint,method,property)`: creates a new adapter with adapter ID, endpoint, method and property of a Web service.

`registerContext(contextid,type,expression,interval, adapterid)`: 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(actionid,url)`: creates a new action with action ID and URL of a Web service.

`registerECA(ecaid,event,condition,action)`: creates a new 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.

`RuCASManager` is deployed as Web service so that various client applications can use RuCAS in a platform-independent manner. Thus, every method of `RuCASManager` is published as Web-API, executed by Web service protocol (SOAP or REST). For example, to create `TempAdapter` in Section 4.1.2, a client invokes Web-API in the following URL form:

```
http://RuCAS/registerAdapter?adapterid=TempAdapter&
endpoint=http://hns/TemperatureSensorService&method
=getTemperature&property=return
```

Using the RuCAS platform, a client application creates 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.

4.3 Implementation

Based on the detailed design, we have implemented the RuCAS platform. For data persistence, we store them in *MongoDB*. The total system comprised of around 4000 lines of code, and the development effort was three man-months. Technologies used for the implementation are as follows: **Language:** Java 1.7.0_21, **Database:** MongoDB 2.4.3, **Web server:** Apache Tomcat 7.0.39, **Web service engine:** Apache Axis2 1.6.2.

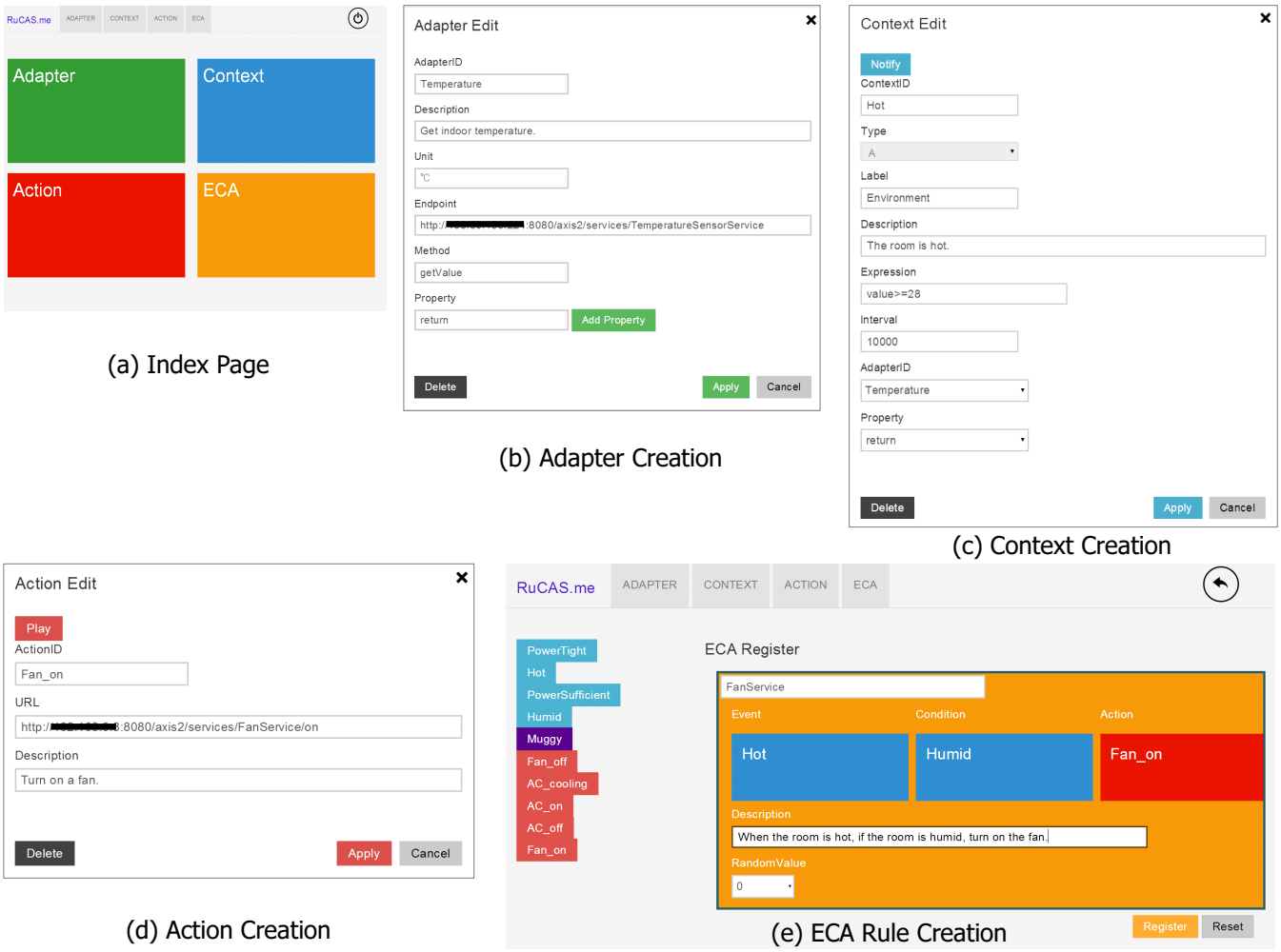

Figure 6: Screenshots of RuCAS.me

5. RUCAS.ME: GUI FRONT-END OF RUCAS

5.1 RuCAS.me

The RuCAS platform provides Web-API for client applications that want to manage custom context-aware services based on RuCAS. However, the Web-API is basically used from applications without human intervention. Thus, the human interface is beyond the scope of the platform.

In order to support users, we have developed a Web application *RuCAS.me*, which is a GUI front-end of the RuCAS platform. Figure 6 shows screenshots of RuCAS.me. Using these screens, users can easily create, edit and delete his/her own RuCAS elements (adapter, context, action, ECA rule) with simple operations on a Web browser.

RuCAS.me was implemented using the following technologies: **Language:** JavaScript, HTML5, **JavaScript Library:** jQuery 2.0.3, **CSS framework:** TwitterBootstrap v3.0.3, bootmetro, **Tested Browser:** Google Chrome 33.0.

5.2 Playing with RuCAS.me

Using Figure 6, we explain how to work with RuCAS.me. Figure 6 (a) shows the index page of RuCAS.me, consisting of four buttons to manage the four RuCAS elements. To support users, some elements are already installed as examples. Combining these preset elements, a user can quickly learn how to create custom context-aware services.

Figure 6 (b) shows an adapter creation page. By filling the form and pressing the apply button, a new adapter is created within the RuCAS platform. Parameters of the form are the ones explained in Section 4.2.1. A created adapter is enumerated in an adapter list page (not shown here due to limited space), where a user can manage the existing adapters.

Figure 6 (c) shows a context creation page. By filling the form and pressing the apply button, a new context is created within the RuCAS platform. Parameters of the form are the ones explained in Section 4.2.2. For the purpose of testing, a user can force any context to be true by the notify button. A created context is enumerated in a context list page (see Figure 7 (a)), where a user can manage the existing contexts. As shown in the figure, a context that currently holds appears as a checked box. This helps a user understand the current situation.

Figure 6 (d) shows an action creation page. By filling the form and pressing the apply button, a new action is created within the RuCAS platform. Parameters of the form are the ones explained in Section 4.2.5. Pressing the play

button executes the action for testing. A created action is enumerated in an action list page (not shown here due to limited space), where a user can manage the existing actions.

Figure 6 (e) shows an ECA rule creation page. The list in the left side of the page enumerates contexts and actions that are already registered in the platform. From the list, a user just selects a preferred context for an event, one or more contexts for a condition, one or more actions. The selected elements appear in the rule pane (in the right side), in form of ECA rule. In the rule pane, clicking an element detaches the element from the ECA rule. In this figure, a user creates an ECA rule to implement a context-aware service *FanService*: "when the room is hot, if the room is humid, turn on the fan". A created ECA rule is enumerated in a ECA list page (see Figure 7 (b)), where a user can manage the existing rules.

6. CASE STUDY

6.1 Sustainable Air-Conditioning Service

To illustrate the practical feasibility of the proposed method, we create an actual practical context-aware service within CS27-HNS using RuCAS platform and RuCAS.me. Here we create a *sustainable air-conditioning service*. This service performs automatic air-conditioning in our laboratory (CS27) when the lab becomes muggy. For this, if the regional power demand (in Kansai area) is sufficient, turn on an air-conditioner. However, if the demand is tight, use a fan instead which consumes much lower energy. Thus, the service contributes to sustainable energy usage.

To implement the service, we use the following distributed Web services:

- **Temperature/Humidity Sensor Services [10]:** Web services that obtain room temperature and humidity of the lab deployed in CS27-HNS.
- **Power Demand API [4]:** External Web service that obtains the current power demand in Kansai region, provided by Yahoo Japan.
- **Appliance Control Service [11]:** Web service that controls appliances in the lab, including the air-conditioner and the fan.

6.2 Creating Service with RuCAS.me

Using RuCAS.me, we create the sustainable air-conditioning service. As mentioned in Section 4.2.7, the service creation is conducted by the four steps.

Step 1: Creating Adapters

We first create three adapters `Temperature`, `Humidity` and `PowerDemand`, using the temperature/humidity sensor services and the power demand API. Parameters for each adapter are summarized in Table 1.

Step 2: Creating Contexts

Using the adapters, we then create five contexts `Hot`, `Humid`, `Muggy`, `PowerSufficient` and `PowerTight`. In this case study, `Hot` (or `Humid`) is defined as a situation that `Temperature` (or `Humidity`) is greater or equal to 28 degrees (or 80 percent, respectively). `Muggy` is defined as a compound context `Hot && Humid`. These three contexts are refreshed every 5 seconds. Using `PowerDemand`, we also create two contexts `PowerSufficient` and `PowerTight`. Here, the threshold of the tight demand is set to 20,000,000 kW, and refresh interval is set to 30 minutes. Parameters for each context are summarized in Table 2. Figure 7 (a) shows RuCAS.me where the five contexts are registered.

Figure 7: List of created contexts and ECA rules

Step 3: Creating Actions

Using the appliance control service, we create three actions `Fan_on` (turn on a fan), `AC_on` (turn on an air-conditioner) and `AC_cooling` (drive an air-conditioner in cooling mode). The parameters are summarized in Table 3.

Step 4: Creating ECA rule

Finally, we create two ECA rules `Sus-AC` and `Sus-Fan` to implement the sustainable air-conditioner service. `Sus-AC` corresponds to the scenario: "when it is muggy in the lab, if the power demand is sufficient, turn on an air-conditioner.". `Sus-Fan` corresponds to the scenario where the demand is tight and the service uses a fan. The parameters for each rule are summarized in Table 4. Figure 7 (b) shows RuCAS.me where the two rules are created.

7. DISCUSSION

7.1 Advantage and Limitations

Using the RuCAS platform, 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 actions (as an autonomous callback of the contexts). This improves the reusability of

Table 1: Parameters for creating adapters

adapterid	endpoint	method	property
Temperature	http://cs27-hns/sensor/temperature	getValue	return
Humidity	http://cs27-hns/sensor/humidity	getValue	return
PowerDemand	http://setsuden.yahooapis.jp/v1/Setsuden	latestPowerUsage?parameter...	{usage, capacity}

Table 2: Parameters for creating contexts

contextid	type	adapter	expression	interval	description
Hot	Atomic	Temperature	value >= 28	5000	It is hot in lab.
Humid	Atomic	Humidity	value >= 80	5000	It is humid in lab.
Muggy	Compound	—	Hot&&Humid	5000	It is muggy (hot and humid) in lab.
PowerSufficient	Atomic	PowerDemand	value < 20000000	1800000	Power demand is sufficient in Kansai region.
PowerTight	Atomic	PowerDemand	value >= 20000000	1800000	Power demand is tight in Kansai region.

Table 3: Parameters for creating actions

actionid	url	description
Fan_on	http://cs27-hns/appliance/fan/on	Turn on a fan.
AC_on	http://cs27-hns/appliance/aircon/on	Turn on an air-conditioner.
AC_cooling	http://cs27-hns/appliance/aircon/cooling	Drive an air-conditioner in cooling mode.

Table 4: Parameters for creating ECA

ecaid	event	condition	action	description
Sus-AC	Muggy	PowerSufficient	{AC_on, AC_cooling}	Sustainable air-conditioning with AC.
Sus-Fan	Muggy	PowerTight	Fan_on	Sustainable air-conditioning with fan.

existing contexts and actions, and enables more flexible creation and management of context-aware services.

To see efficiency of service creation, we measured the time taken for a graduate student to create the sustainable air-conditioning service using RuCAS.me. It was shown that the graduate students spent less than ten minutes to complete the task. Although more detailed evaluation is needed, RuCAS seems to work quite efficiently, to implement custom context-aware services with various Web services. More detailed evaluation with non-expert users who are not familiar with Web services programming is left for our future work.

A limitation of the current implementation is the *expressive power* of the context expression. To avoid complexity, we define every context expression as a simple logical expression over *present values* of Web services. More complex contexts with past or future (estimated) values combined with temporal logic are beyond the scope. This is our design choice, counting the trade-off between the expressiveness and complexity. A solution to deal with such a complex context is to implement it as an external Web service that returns the evaluation result. Then, RuCAS can *import* the Web service via an adapter, to use the complex context for ECA rules.

Another limitation is that RuCAS cannot validate *semantic consistency* of the services. Non-expert users may mistakenly create ECA rules which contradict to their intention (e.g., when it is hot, turn on a heater). A method that supports the semantic validation, debugging and testing would be helpful for users to create reliable services.

Furthermore, creating a number of ECA rules may cause functional conflicts among the rules. This is known as the *feature interaction problem* [18]. We previously proposed a method to manage this problem [8]. We plan to integrate the method with the RuCAS platform in future development.

7.2 Related Work

Sheng et al. proposed *ContextServ* [14], a platform for rapid development of context-aware Web services. This method defines every context in a UML-based language (ContextUML), then translates the definition into a Web service. Unlike RuCAS, the method is supposed to be used by professional system developers who can understand and write UML. Although the expressiveness is higher than RuCAS, the method requires high expertise to use. We consider that the method suits well for implementing the Web services for complex contexts, Which are discussed in Section 7.1.

Rasch et al. proposed a context-driven personalized service discovery system [13]. Niu et al. proposed CARSA [12], a context-aware reasoning-based service agent model for AI planning of Web service composition. These studies focus on an accuracy of the context-aware service discovery and composition, whereas RuCAS aims the systematic creation and management of custom context-aware services by various clients. Hence, the targets are different. These methods might be used in RuCAS to enable *automatic* creation and management of ECA rules.

Several Web services for managing custom context-aware services come onto the market. *IFTTT* [1] is a Web service that coordinates various network services (e.g., Gmail, Twitter, RSS feeds, etc.) based on a rule (called recipe) of "if this then that". *WigWag* [2] provides a similar but more device-centric service. It defines custom context-aware services based on "when then" logic over proprietary sensors and control devices. These services basically use *ready-made* data source (called channel) to define events and actions. RuCAS differs in using custom-made data sources by creating adapters for any Web services. This enables more generic and a wider range of contexts and actions. Also as for the rule description logic, IFTTT and WigWag basically use an event and an action only, while our ECA rule uses a condi-

tion in addition to them. The condition allows a service to have different actions for the same event.

Xively [3] is a cloud service that provides a data platform for the internet of things. By binding a real-world device with Xively API, data from the device can be shared within the cloud service. Xively mainly focuses on gathering and sharing data from devices in global locations, rather than defining and reusing custom contexts and actions. Therefore, we consider that RuCAS can complement Xively by using Xively as data source of RuCAS.

8. CONCLUSION

In this paper, we have described the detail of the design and the implementation of RuCAS and its service platform, called RuCAS platform. RuCAS is a rule-based framework for creating and managing context-aware services with Web services. In the RuCAS, contexts, actions and services are systematically managed by five layers: Web service layer, adapter layer, context layer, action layer and ECA rule layer. We also developed RuCAS.me, which is a GUI front-end of the RuCAS platform. A case study demonstrated the practical feasibility. Using the RuCAS platform and RuCAS.me, users and client applications can manage their own context-aware services efficiently and flexibly.

Our future work includes experimental evaluation with non-expert users, as well as investigation of semantic validation and the feature interaction problem. Also, we are interested in finding new application domains of RuCAS, such as factory controls, smart cities and cloud service management.

9. ACKNOWLEDGMENTS

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), (B) (No.26280115), Young Scientists (B) (No.26730155)] and Kawanishi Memorial ShinMaywa Education Foundation.

10. REFERENCES

[1] IFTTT. `https://ifttt.com`. Accessed: 2014-07-30.

[2] Wigwag. `http://www.wigwag.com`. Accessed: 2014-07-30.

[3] Xively. `https://xively.com`. Accessed: 2014-07-30.

[4] Yahoo JAPAN Web API. `http://developer.yahoo.co.jp/webapi/shinsai`. Accessed: 2014-07-30.

[5] Y. Chon and H. Cha. Lifemap: A smartphone-based context provider for location-based services. *Transactions on Pervasive Computing*, 10(2):58–67, 2011.

[6] N. Cohen, J. Black, P. Castro, M. Ebling, B. Leiba, A. Misra, and W. Segmuller. Building context-aware applications with context weaver. IBM Research Division, 2004.

[7] X. Li and W. Zhang. The design and implementation of home network system using OSGi compliant middleware. *Transactions on Consumer Electronics*, 50(2):528–534, 2004.

[8] M. Nakamura, H. Igaki, Y. Yoshimura, and K. Ikegami. Considering online feature interaction detection and resolution for integrated services in home network system. In *International Conference on Feature Interactions in Telecommunications and Software Systems*, pages 191–206, 2009.

[9] M. Nakamura, S. Matsuo, and S. Matsumoto. Supporting end-user development of context-aware services in home network system. In R. Lee, editor, *Studies in Computational Intelligence*, pages 159–170. Springer, 2012.

[10] M. Nakamura, S. Matsuo, S. Matsumoto, H. Sakamoto, and H. Igaki. Application framework for efficient development of sensor as a service for home network system. In *International Conference on Services Computing*, pages 576–583, 2011.

[11] M. Nakamura, A. Tanaka, H. Igaki, H. Tamada, and K. Matsumoto. Constructing home network systems and integrated services using legacy home appliances and Web services. *International Journal of Web Services Research*, 5(1):82–98, 2008.

[12] W. Niu, G. Li, H. Tang, X. Zhou, and Z. Shi. CARSA: A context-aware reasoning-based service agent model for AI planning of Web service composition. *Journal of Network and Computer Applications*, 34(5):1757–1770, 2011.

[13] K. Rasch, F. Li, S. Sehic, R. Ayani, and S. Dustdar. Context-driven personalized service discovery in pervasive environments. *World Wide Web*, 14(4):295–319, 2011.

[14] Q. Sheng, S. Pohlenz, J. Yu, H. Wong, A. H. H. Ngu, and Z. Maamar. ContextServ: A platform for rapid and flexible development of context-aware web services. In *IEEE 31st International Conference on Software Engineering*, pages 619–622, 2009.

[15] H. Takatsuka, M. Nakamura, S. Saiki, and S. Matsumoto. Developing service platform for web context-aware services towards self-managing ecosystem. In *The Third International Workshop on Self-Managing Pervasive Service Systems*, 2014. (to appear).

[16] H. Takatsuka, M. Nakamura, S. Saiki, and S. Matsumoto. A rule-based framework for managing context-aware services based on heterogeneous and distributed Web services. In *15th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*. IEEE Computer Society, 2014.

[17] T. Velte, A. Velte, and R. Elsenpeter. *Cloud Computing, A Practical Approach*. McGraw-Hill, Inc., 1st edition, 2010.

[18] M. Wilson, M. Kolberg, and E. Magill. Considering side effects in service interactions in home automation-an online approach. *Feature Interactions in Software and Communication Systems IX*, pages 172–187, 2008.

[19] G. Wu, S. Talwar, K. Johnsson, N. Himayat, and K. Johnson. M2M: From mobile to embedded internet. *IEEE Communications Magazine*, 49(4):36–43, 2011.

[20] S. Yamamoto, N. Kouyama, K. Yasumoto, and M. Ito. Maximizing users comfort levels through user preference estimation in public smartspaces. In *International Conference on Pervasive Computing and Communications Workshops*, pages 572–577, 2011.