# Developing Service Platform for Web Context-Aware Services Towards Self-Managing Ecosystem

Hiroki Takatsuka(✉), Sachio Saiki, Shinsuke Matsumoto, and Masahide Nakamura

Graduate School of System Informatics, Kobe University, Kobe, Japan
tktk@ws.cs.kobe-u.ac.jp, sachio@carp.kobe-u.ac.jp,
{shinsuke,masa-n}@cs.kobe-u.ac.jp

**Abstract.** The convergence of cloud/service computing and M2M/IoT systems provides real-world sensing and actuation as globally distributed Web services. Context-aware services using such Web services (we call them *Web Context-Aware Services, Web-CAS*) are promising in many systems. However, definition of contexts and Web services to be used highly depend on individual environments and preferences. Therefore, it is essential to have a place for self-management, where individual users can efficiently manage their own Web-CAS by themselves. In this paper, we develop a service platform, called *RuCAS platform*, which works as PaaS for self-managing Web-CAS. In the platform, contexts and actions are defined by adapting the distributed Web services, and every Web-CAS is managed in form of an *ECA (Event-Condition-Action) rule*. Through Web-API of RuCAS, individual clients can rapidly create, update, delete and execute custom contexts and services. To support non-expert users, we implement a GUI front-end of the RuCAS platform, called *RuCAS.me*. A case study of sustainable air-conditioning demonstrates practical feasibility. Finally, we discuss how the RuCAS platform works to achieve self-managing ecosystem of Web-CAS.

**Keywords:** Web services · Context-awareness · Self-management · Event-condition-action rule · Home network system

## 1 Introduction

A *context* refers to situational information derived from dynamic data of a sensor or a system. A *context-aware service* is a service that automatically detects a change of contexts and performs appropriate actions corresponding to the context change [7]. 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, such context-aware services had been studied extensively in ubiquitous/pervasive computing,


F. Toumani et al. (Eds.): ICSOC 2014, LNCS 8954, pp. 270–280, 2015.
DOI: 10.1007/978-3-319-22885-3_24

where each contexts was defined using situated sensors [15] or hand-held devices (e.g., smartphone) within a local smart space [4].

However, cloud/service computing [13] and IoT/M2M technologies [14] dramatically extend the scope of context-aware services. Cloud/service computing abstracts heterogeneous computing resources and data, and provides them as interoperable Web services. IoT/M2M allow various devices to communicate with each other without human intervention. The combination of both technologies provides real-world sensing and actuation as globally distributed Web services. Relevant studies include *LinkSmart middleware* [6], *service-oriented home network system* [10], and *sensor service framework* [9]. Using such Web services to build context-aware services is a promising approach, since a wide variety of contexts and actions can be seamlessly defined over distributed and heterogeneous resources. In this paper, we refer to such context-aware services with Web services as *Web context-aware services* (or simply *Web-CAS*).

A major challenge of Web-CAS lies in how to manage unstable and complex relations among Web services, defined contexts, and actions caused by the contexts. In general, definition of contexts and Web services to be used highly depend on individual environments and preferences. For instance, in the Automatic Air-conditioning service, which temperature sensor and air-conditioner should be used depends on the room where the service is operated. The definition of `Hot` context with 28 degree may not be reasonable in winter. A user may prefer to turn on a fan instead of the expensive air-conditioner. To meet various requirements and preferences, every Web-CAS should not be hard-coded. Instead, it is essential to have a place for self-management, where individual users can efficiently manage own Web-CAS by themselves.

In this paper, we develop a service platform, called *RuCAS platform*. Intuitively, it works as PaaS (Platform-as-a-Service), which provides self-managing Web-CAS capabilities for various clients. The RuCAS platform is designed specifically based on the following three requirements: **(R1)** every context-aware service can be defined by arbitrary Web services in an intuitive and systematic manner, **(R2)** individual client can dynamically create, update, delete and execute custom contexts, actions and services, **(R3)** even non-expert users can develop and manage their own context-aware services. As far as we know, there exists no service platform satisfying all the requirements.

To satisfy requirement R1, RuCAS manages every Web-CAS in form of an *ECA (Event-Condition-Action) rule*. The event is a context that triggers a service. The condition refers to a guard condition to execute the service. The action is a Web service executed by the service. As for requirement R2, we implement the platform with five layers: *Web service layer, adapter layer, context layer, action layer* and *ECA rule layer*. Each layer exhibits Web-API (REST or SOAP) so that individual clients can manage their own elements. Finally, to meet requirement R3, we implement a GUI front-end of the 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.

To evaluate practical feasibility, we conduct a case study using the RuCAS platform and RuCAS.me. Integrating an external Web service with our home network system [10], we implement a sustainable air-conditioning service, which contributes to peak shaving of regional energy consumption. We also discuss how the RuCAS platform works to achieve *self-managing ecosystem* of Web-CAS.

## 2 RuCAS: Rule-Based Management Framework for Web Context-Aware Services

*RuCAS (Rule-based management framework for Web Context-Aware Service)* is a service framework, specifically designed to help individual clients (human users and software applications) to easily create and manage their own Web-CAS.

### 2.1 Event-Condition-Action (ECA) Rule

The *ECA rule* is an important design decision of RuCAS, which defines every Web-CAS as a triplet of [Event, Condition, Action]. This design decision is to implement Web-CAS by *loose coupling* of Web services as data sources, contexts defined with the data sources, and actions to be performed by the contexts.

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*. However, the above rule lacks flexibility, because 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*. More specifically,

- 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 as a conjunction of one or more contexts.
- An *action* is a set of operations executed by a context-aware service. An action is defined by one or more Web services.
- *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.

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: $[E : Hot,\ C : \{PresentUser\},\ A : \{AC.on\}]$. Thus, the ECA rules give an intuitive but systematic foundation to define Web-CAS, which satisfies the requirement R1.

### 2.2 RuCAS Platform: RuCAS as Service Platform

To allow various clients to dynamically manage their own contexts, actions and ECA rules via network, we implement RuCAS as a service platform. The implementation is deployed in a cloud as PaaS (Platform-as-a-Service).

Figure 1 represents a system architecture of the RuCAS platform. To build ECA rules from loosely-coupled components, the platform consists of five layers: Web service layer, adapter layer, context layer, action layer and ECA rule layer.

**Web Service Layer:** This layer contains existing Web services used as *input* or *output* of Web-CAS. 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 a value from a sensor service, status of a device, dynamic Web information (e.g., weather, stock price), RSS, 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 (e.g., send an email, post a comment to SNS, etc.).

**Adapter Layer:** To obtain data from an input Web service, a client needs to invoke Web-API and extract necessary data by parsing the return value. However, Web-API and the return value vary from one service to another. Hence, this layer creates an *adapter* that normalizes the heterogeneous interface. Specifically, every Web-API used to obtain data is adapted to `getValue()`. For example, we can create `TempAdapter`, with a temperature Web-API, say http://www.hns/TemperatureSensorService/getTemperature. RuCAS adapts the Web-API so that `TempAdapter.getValue()` returns the temperature.

**Context Layer:** This context layer manages contexts defined by data from Web services via the adapter layer. Every context is defined by *context ID* and *context*

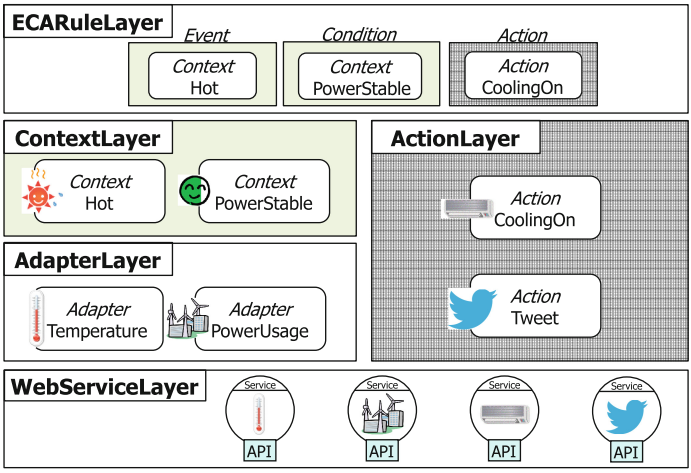


**Fig. 1.** Architecture of RuCAS platform

*expression*. The context ID is a label to identify every context. The context expression is a logical formula, in the form of `Adapter.value comp_op const`, where `comp_op` is a comparison operator and `const` is a constant value. For example, to define `Hot` context to be "the temperature is more than 28 degrees", RuCAS describes it by `[Hot: TempAdapter.value > 28]`. Similarly, to define `Humid` to be "the humidity is 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 `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]`.

**Action Layer:** This layer manages actions, each of which wraps an output Web service. An action 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 create an action `CoolingOn`, by using an air-conditioner service, say http://www.hns/ACService/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 context-aware services as ECA rules. 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 rules are *executed* based on the semantics (see Sect. 2.1).

To meet the requirement R2, each layer exhibits Web-API to create, update, delete and execute the custom elements. Using REST or SOAP protocol, clients in various platforms can execute the Web-API to self-manage their Web-CAS.

The RuCAS platform was implemented with the following technologies: **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.

### 2.3 RuCAS.me

We have also developed a Web application, called *RuCAS.me*, to support non-expert users who are unfamiliar with Web service programming (see the requirement R3). RuCAS.me works as a GUI front-end of the RuCAS platform.

Figure 2 shows screenshots, with which a user can easily create, edit and delete own RuCAS elements (adapter, context, action, ECA rule) using a Web browser. Figure 2 (a) shows the index page of RuCAS.me consisting of four


**Fig. 2.** Screenshots of RuCAS.me

buttons to manage the four elements. Figure 2 (b) shows an adapter creation page. By filling the form and pressing the apply button, a new adapter is created within the RuCAS platform. Figure 2 (c) shows a context creation page. By filling the form and pressing the apply button, a new context is created within the RuCAS platform. A created context is enumerated in a context list page (see Fig. 2 (f)), 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 2 (d) shows an action creation page. By filling the form and pressing the apply button, a new action is created within the RuCAS platform.

Figure 2 (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 and one or more actions. The selected elements appear in the rule pane (in the right side), in the form of ECA rule. In this figure, a user creates an ECA rule to implement a context-aware service *FanService*: “when

**Table 1.** Parameters for creating adapters

| Adapterid | Endpoint | Method | Property |
|---|---|---|---|
| Temperature | http://www.cs27-hns/sensor/temperature | getValue | return |
| Humidity | http://www.cs27-hns/sensor/humidity | getValue | return |
| PowerDemand | http://setsuden.yahooapis.jp/Setsuden | latestPower Usage | {usage,capacity} |

**Table 2.** Parameters for creating contexts

| Contextid | Type | Adapter | Expression | Interval | Description |
|---|---|---|---|---|---|
| Hot | A | Temperature | value>=28 | 5000 | It is hot in lab |
| Humid | A | Humidity | value>=80 | 5000 | It is humid in lab |
| Muggy | C | — | Hot&&Humid | 5000 | It is muggy in lab |
| PowerSufficient | A | PowerDemand | value<20000000 | 1800000 | Power demand is sufficient in Kansai region |
| PowerTight | A | PowerDemand | value>=20000000 | 1800000 | Power demand is tight in Kansai region |

**Table 3.** Parameters for creating actions

| Actionid | url | Description |
|---|---|---|
| Fan_on | http://www.cs27-hns/appliance/fan/on | Turn on a fan |
| AC_on | http://www.cs27-hns/appliance/AC/on | Turn on an AC |
| AC_cooling | http://www.cs27-hns/appliance/AC/cooling | Drive an AC in cooling mode |

the room is hot, if the room is humid, turn on the fan". A created ECA rule is enumerated in an ECA list page (see Fig. 2 (g)) to manage existing rules.

RuCAS.me was implemented with 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.

## 3 Case Study: Sustainable Air-Conditioning Service

To illustrate the practical feasibility of the developed system, we create the *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 that consumes much lower energy. To implement the service, we use the following Web services:

- **Temperature/Humidity Sensor Services** [9]**:** Web services that obtain room temperature and humidity of in CS27.
- **Power Demand API** [3]**:** External Web service that obtains the current power demand in Japan Kansai region, provided by Yahoo Japan.
- **Appliance Control Service** [10]**:** Web service that controls appliances in the lab, including the air-conditioner and the fan.

**Table 4.** Parameters for creating ECA

| ecaid | Event | Condition | Action | Description |
|---|---|---|---|---|
| Sus_AC | Muggy | PowerSufficient | {AC_on, AC_cooling} | Air-conditioning with an AC |
| Sus_Fan | Muggy | PowerTight | Fan_on | Air-conditioning with a fan |

Using RuCAS.me, we create the service based on the following recipe:

**Step 1 (Creating Adapters):** We first create three adapters `Temperature`, `Humidity` and `PowerDemand`, using the temperature/humidity sensor services and the power demand API. The parameters 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 min. Parameters for each context are summarized in Table 2. Figure 2 (f) shows RuCAS.me where the five contexts are registered.

**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), as shown 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 2 (g) shows the two rules are created.

## 4 Discussion

### 4.1 Operating RuCAS Platform for Self-Managing Ecosystem

The RuCAS platform can be a key component for *self-managing ecosystem*, which alleviates increasing complexity, scale and development/operation cost of Web-CAS. Figure 3 shows a block diagram involving the RuCAS platform and related components. A solid arrow represents a manual (or proprietary) operation performed by a user, while a dotted arrow represents an autonomic operation.

First, individual users create custom contexts and ECA rules (with RuCAS.me or proprietary client software). For given rules, the RuCAS platform periodically pulls current status from distributed Web services, and actuates designated Web services. This forms a small ecosystem as depicted by a

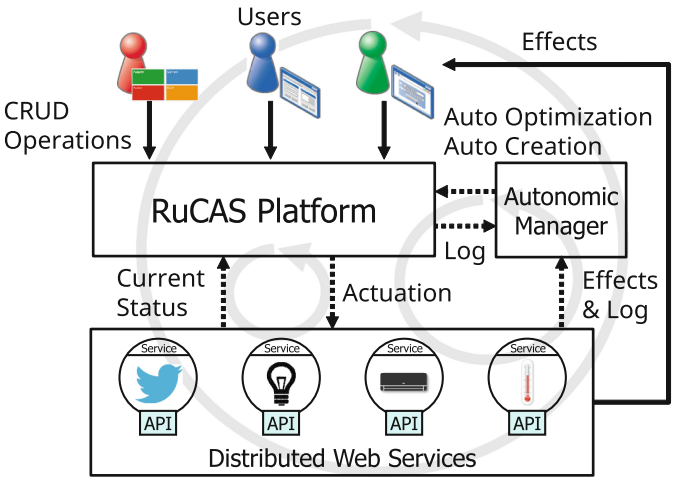


**Fig. 3.** RuCAS platform as component of self-management ecosystem

left-small circle. The actuation of the Web service yields some *effects* within the global/local environment. The users monitor the effects, and update contexts and rules if needed. This forms a global ecosystem depicted by a large circle.

It is also promising to integrate an external *autonomic manager*, which conducts autonomic creation and optimization of contexts and ECA rules. This causes an extra ecosystem depicted as a right-small circle in Fig. 3. The integration is quite easy, since the RuCAS platform is interoperable with any other system via Web-API. We are currently developing an autonomic manager for our home network system with referring to related studies (e.g., [5,16]).

### 4.2 Related Work

As for self-managing pervasive systems, Zhang et al. proposed a semantic web based approach [16], and Ada et al. [5] proposed extension of iPOJO. They aim to satisfy four aspects of self-management [8] (i.e., self-configuration, self-optimization, self-healing and self-protection), by managing all pervasive objects under a proprietary middleware. On the other hand, RuCAS coordinates existing distributed Web services, for which we cannot enforce a specific middleware.

Several studies of context-awareness with Web services exist. Rasch et al. proposed a context-driven personalized service discovery system [12]. Niu et al. proposed *CARSA* [11], a context-aware AI planning of Web service composition. These studies use contexts to improve an accuracy of Web service discovery and composition. Whereas, RuCAS aims the systematic self-management of custom context-aware services using Web services. Thus, the targets are different.

Practical services for self-managing context-aware services recently come onto the market. *IFTTT* [1] coordinates various network services (e.g., Gmail, Twitter, RSS feeds, etc.) based on a rule of "if this then that". *WigWag* [2] 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 data sources by creating adapters for arbitrary Web services. Also IFTTT and WigWag basically use an event and an action only, while our ECA rule uses a condition together with them. This makes RuCAS more expressive.

## 5 Conclusion

In this paper, we have developed the RuCAS platform for self-managing context-aware services with distributed Web services (Web-CAS). In the platform, contexts, actions and services are systematically managed by five layers: Web service, adapter, context, action and ECA rule. Using Web-API, individual clients can manage their own context-aware services efficiently and flexibly. To support non-expert users, We also developed a GUI front-end, RuCAS.me. A case study demonstrated the practical feasibility. Finally, we discussed how the RuCAS platform work within the self-managing ecosystem of Web-CAS.

Our future work includes development of the autonomic manager discussed in Sect. 4.1, investigation of self-healing and self-protection aspects of Web-CAS.

**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.

## References

1. IFTTT. https://ifttt.com. Accessed 30 July 2014
2. Wigwag. http://www.wigwag.com. Accessed 30 July 2014
3. Yahoo JAPAN Web API. http://developer.yahoo.co.jp/webapi/shinsai. Accessed 30 July 2014
4. Chon, Y., Cha, H.: Lifemap: a smartphone-based context provider for location-based services. Trans. Pervasive Comput. **10**(2), 58–67 (2011)
5. Diaconescu, A., Bourcier, J., Escoffier, C.: Autonomic iPOJO: towards self-managing middleware for ubiquitous systems. In: IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, pp. 472–477 (2008)
6. Eisenhauer, M., Rosengren, P., Antolin, P.: Hydra: a development platform for integrating wireless devices and sensors into ambient intelligence systems. In: Giusto, D., Iera, A., Morabito, G., Atzori, L. (eds.) The Internet of Things, pp. 367–373. Springer, New York (2010)
7. Gu, T., Pung, H.K., Zhang, D.Q.: A service-oriented middleware for building context-aware services. J. Netw. Comput. Appl. **28**(1), 1–18 (2005)
8. Kephart, J., Chess, D.: The vision of autonomic computing. Computer **36**(1), 41–50 (2003)
9. Nakamura, M., Matsuo, S., Matsumoto, S., Sakamoto, H., Igaki, H.: Application framework for efficient development of sensor as a service for home network system. In: International Conference on Services Computing, pp. 576–583 (2011)

10. Nakamura, M., Tanaka, A., Igaki, H., Tamada, H., Matsumoto, K.: Constructing home network systems and integrated services using legacy home appliances and Web services. Int. J. Web Serv. Res. **5**(1), 82–98 (2008)
11. Niu, W., Li, G., Tang, H., Zhou, X., Shi, Z.: CARSA: a context-aware reasoning-based service agent model for AI planning of Web service composition. J. Netw. Comput. Appl. **34**(5), 1757–1770 (2011)
12. Rasch, K., Li, F., Sehic, S., Ayani, R., Dustdar, S.: Context-driven personalized service discovery in pervasive environments. World Wide Web **14**(4), 295–319 (2011)
13. Velte, T., Velte, A., Elsenpeter, R.: Cloud Computing, A Practical Approach, 1st edn. McGraw-Hill Inc, New York (2010)
14. Wu, G., Talwar, S., Johnsson, K., Himayat, N., Johnson, K.: M2M: from mobile to embedded internet. IEEE Commun. Mag. **49**(4), 36–43 (2011)
15. Yamamoto, S., Kouyama, N., Yasumoto, K., Ito, M.: Maximizing users comfort levels through user preference estimation in public smartspaces. In: International Conference on Pervasive Computing and Communications Workshops, pp. 572–577 (2011)
16. Zhang, W., Hansen, K.: Semantic web based self-management for a pervasive service middleware. In: Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems, pp. 245–254 (2008)