KULOCS: unified locating service for efficient development of location-based applications

Hiroki Takatsuka Seiki Tokunaga Sachio Saiki Shinsuke Matsumoto Masahide Nakamura

Received 7 September 2015
Revised 6 October 2015
Accepted 1 February 2016

KULOCS: unified locating service for efficient development of location-based applications

Hiroki Takatsuka, Seiki Tokunaga, Sachio Saiki, Shinsuke Matsumoto and Masahide Nakamura
Graduate School of System Informatics, Kobe University, Kobe, Japan

Abstract

Purpose – The purpose of this paper is to develop a facade for seamlessly using locating services and enabling easy development of an application with indoor and outdoor location information without being aware of the difference of individual services. To achieve this purpose, in this paper, a unified locating service, called KULOCS (*Kobe-University Unified LOCating Service*), which horizontally integrates the heterogeneous locating services, is proposed.

Design/methodology/approach – By focusing on technology-independent elements [when], [where] and [who] in location queries, KULOCS integrates data and operations of the existing locating services. In the data integration, a method where the time representation, the locations and the namespace are consolidated by the Unix time, the location labels and the alias table, respectively, is proposed. Based on the possible combinations of the three elements, an application-neutral application programming interface (API) for the operation integration is derived.

Findings – Using KULOCS, various practical services are enabled. In addition, the experimental evaluation shows the practical feasibility by comparing cases with or without KULOCS. The result shows that KULOCS reduces the effort of application development, especially when the number of locating services becomes large.

Originality/value – KULOCS works as a seamless facade with the underlying locating services, the users and applications consume location information easily and efficiently, without knowing concrete services actually locating target objects.

Keywords Web services, Location information, Locating service, Location-aware, Positioning system

Paper type Research paper

1. Introduction

Smart combination of internet of things (IoT) (Vermesan *et al.*, 2011), positioning systems and cloud services enables a sophisticated platform to acquire and manage *locations* of mobile users and objects. Nowadays, every smartphone is equipped with global positioning system (GPS). Also, various GPS modules for IoT have appeared on the market (e.g. OriginGPS and TinyGPS). The latest *indoor positioning systems* (*IPS*) can locate users even inside buildings or underground, where GPS cannot cover. The enabling technologies of IPS include Wi-Fi (e.g. Skyhook[1]), Bluetooth beacons (Kohne and Sieck, 2014), radio-frequency identification (RFID) (Ting *et al.*, 2011), pedestrian

This research was partially supported by the Japan Ministry of Education, Science, Sports and Culture [Grant-in-Aid for Scientific Research (B) (No.26280115, No.15H02701), Young Scientists (B) (No.26730155) and Challenging Exploratory Research (15K12020)].

International Journal of Pervasive Computing and Communications
Vol. 12 No. 1, 2016
pp. 154-172

1742-7371
DOI 10.1108/IJPCC-01-2016-0004

dead reckoning (Pratama *et al.*, 2012) and indoor messaging system (IMES) (Manandhar and Torimoto, 2011). Gathering such indoor/outdoor location information in the cloud would create a great variety of location-based services and applications.

The location information gathered in the cloud should be provided *as a service*, so that client applications can easily consume the locations based on standard Web service protocols. We call such a cloud service as *locating service* in this paper. In fact, several practical services have come in the market recently. They include Swarm[2], Glympse[3], Google Maps application programming interfaces (APIs)[4], Pathshare[5], Apple Family Sharing[6] and IndoorAtlas[7]. Although features and operation policies vary from one service to another, the basic idea is to use the cloud for exchanging or sharing location information acquired by a certain positioning system. Most services provide Web-API for application developers.

In general, there is no compatibility among different locating services and API, as they are individually developed and operated. Each service is tightly coupled with the underlying positioning system. For example, Glympse assumes using GPS information collected by smartphones, while IndoorAtlas uses magnetic field to locate the position inside a building. Thus, Glympse cannot directly use the data of IndoorAtlas and vice versa. To cover both indoor and outdoor locations, one may want to *integrate* these two services. However, the lack of compatibility forces the application developer to use different API, and to perform expensive data integration within the application.

Figure 1 shows the conventional architecture to integrate the existing locating services. Let us assume an application, say "where-are-you?" with which a user *A* tries to find location of another mobile user *B*. Suppose also that *B* is in either indoor or outdoor space, and is located by a certain locating service. When *A* executes a query "Where is *B*?" the application has to invoke all possible locating services to find *B*. Although the query "Where is *B*?" is essentially simple, the application has to know how to invoke API and interpret the result for every locating service. This makes the application complex, low-performing and non-scalable.

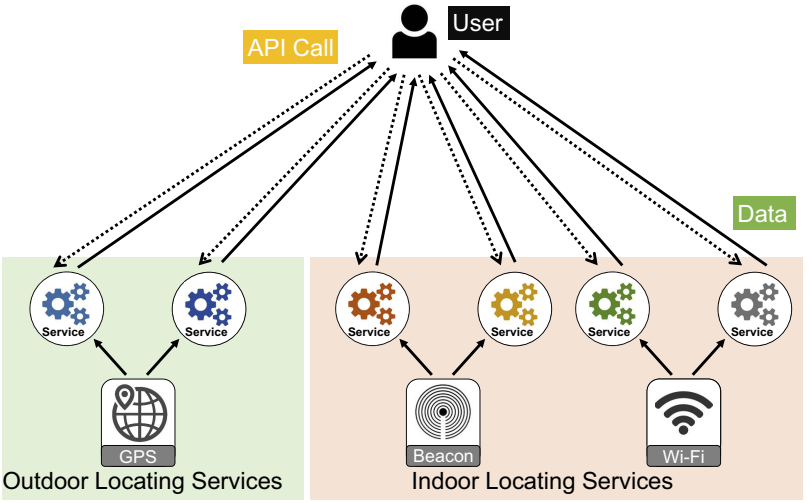

Figure 1. Conventional architecture to integrate locating services

To cope with the problem, in this paper, we propose a unified locating service, called *KULOCS* (*Kobe-University Unified LOCating Service*). KULOCS horizontally integrates the existing heterogeneous locating services, and provides an abstraction layer between the applications and the locating services. To make location queries compatible among many locating services, we design KULOCS with three technology-independent elements [when], [where] and [who].

Based on the three elements, KULOCS integrates data and operations of the heterogeneous locating services. In the data integration, we propose a method where different representation of time, heterogeneous locations and different namespace of users are consolidated by *Unix time*, *location labels* and *alias table*, respectively. The location labels consist of local label and global label, which abstract concrete coordinates of IPS and GPS, respectively. A KULOCS user queries every location by a label, whereas KULOCS internally converts the label to a specific representation for individual locating services.

For the operation integration, we propose KULOCS-API, which integrates heterogeneous operations by possible combinations of [when], [where] and [who]. The API is deployed as a Web service, so that applications on various platform can easily consume KULOCS. For example, the query "Where is *B*?" of "where-are-you?" is simply implemented by `http://kulocs/where?user=B&time=now`. For this, the application need not know how *B* is located by which service. Thus, the application can consume loction quite easily and efficiently.

In this paper, we also design and implement the proposed KULOCS as a Java Web service. The current version supports the integration of the following locating services: a GPS-based outdoor locating service and a BLE (Bluetooth Low Energy)-based indoor locating service. On top of KULOCS implemented, we develop two application services. The first service is *Umbrella Reminder Service*, which prompts a user to take an umbrella when it is raining. This service uses KULOCS to evaluate a location context "when a user leaves home", is defined by the position of the user. The other service is *Stay Areas Visualization Service*, which displays the history of areas where users have visited on a given day.

To evaluate practical feasibility, we conduct the experiment, which compares application development with KULOCS. Specifically, we implement two different versions of the same application, where one is with KULOCS and another is without. The two versions are examined from the perspective of the lines of code and response time. We also conduct the performance evaluation of KULOCS-API, where different methods to obtain the same information (e.g. "Is user tktk in Kobe University now?") are compared. Finally, we investigate other promising services that KULOCS makes feasible.

The original version of this paper has been published as a conference paper accepted in the international conference iiWAS2015 (Takatsuka *et al.*, 2015). Based on comments and discussion we received in the conference, we extensively revise the paper for this journal paper. The most significant update is the addition of Section 6, where we discuss pragmatic issues expected when providing KULOCS as a practical service. They include stakeholders, authentication, security and privacy issues. We believe that those changes will help readers fully understand the integrating locating services, and develop similar systems, efficiently.

2. KULOCS (Kobe-University Unified LOCating Service)

2.1 Overview

In this section, we explain the basic principles of *KULOCS* (*Kobe-University Unified LOCating Service*). Figure 2 shows the architecture. KULOCS works as a *facade* of the heterogeneous locating services. It provides the unified interface (KULOCS-API) for a user, by which the user can access to different locating services seamlessly, without being aware of the difference of individual services. Because KULOCS is an abstract layer that integrates heterogeneous locating services, we have to achieve the following issues:

- *Data Integration*: Individual locating services represent location information in different ways. Hence, KULOCS must exploit unified location data representation that is independent of any specific service or positioning system.
- *Operation Integration*: Individual locating services exhibit own operations in terms of API, which vary from a service to another. KULOCS needs to integrate them and provide generic API (i.e. KULOCS-API) to a user.

Our key idea to achieve the above integration is to focus the following technology-independent elements, which are necessary for any service to locate an object:

- *When*: Represents the date and time when the target object exists.
- *Where*: Represents the location where the target object exists.
- *Who*: Represents the identity of the target object.

Note that other interrogatives like how, what and why are not included because they tend to be technology-oriented. KULOCS is designed to accept *generic* queries based on possible combinations of the above three elements. KULOCS then translates the generic query to service-specific queries for individual services.

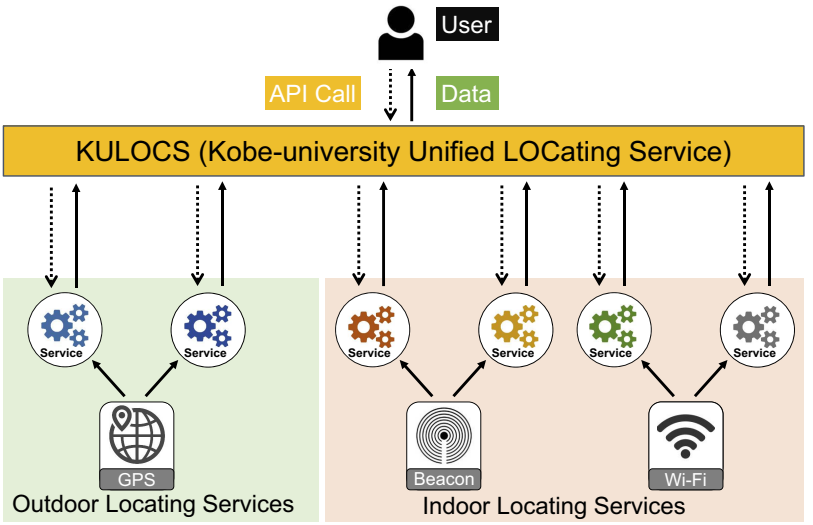

Figure 2.
Architecture of KULOCS

2.2 Data integration

We here describe how to integrate location data of heterogeneous locating services. To help to understand, let us consider the following data records.

- *L1*: {time:2015-06-21T08:50:12+0900, user:tktk, location: {latitude:35.4313, longitude:135.147, address:"1-1 Rokkodai Nada Kobe Japan"}}
- *L2*: Takatsuka is now in (3.0, 4.5, 0.5) from entrance of ShopABC.
- *L3*: Mon Jun 29 15:49:34 CEST 2015, Object123, KobeUniv.Lab.S101

L1 describes a location of user tktk by a geographic coordinate, where we imagine the data are taken by a GPS-based service. L2 would be obtained by a fine-resolution IPS, which represents the current position of Takatsuka by 3D offset from a reference point. L3 describes that Object123 is in room S101 of our laboratory, which may be located by a certain zone-based IPS. Note that L1, L2 and L3 use different time representations (and time zones).

To integrate these heterogeneous location data, we consider the elements [when], [where] and [who]. As for [when], it is easy to introduce the common representation with the *Unix timestamp*, which is the number of seconds elapsed from January 1, 1970 at UTC. KULOCS deals with any time information by the Unix time.

As for [where], there are many ways and different granularity levels to represent a location. The GPS coordinate looks for generic representation that can describe exact locations. However, it is too detailed for a user to specify it as a parameter of location queries. Also, the GPS coordinate is not useful for indoor locations, which are often relative coordinates from the reference point.

To compromise different granularity levels and various use cases, we propose to represent every location by a *location label*. A location label is a unique string that is bound for a location information. Just for convenience, we introduce two kinds of labels: *local label* and *global label*. The local label is a string, written in `position@building`, to be used to represent an indoor location. In the string, `building` represents the ID of a building, and `position` represents the name of the position in the building. For example, a local label `casher@ShopABC` is used to refer to the location in L2. On the other hand, the global label is a string without @, to be used to represent an outdoor location. For example, we can bind a global label `kobe_univ` to the location in L1.

Thus, KULOCS represents every location by a location label. It internally maintains binding between a label and actual location information with the *location table* shown in Table I. We assume that the location labels are registered in the table by users in a crowd-sourcing fashion, and shared among the users.

Finally, as for [who], because every locating service has a different namespace for users and objects, KULOCS has an alias table, which consolidates different IDs for the same user (or object) into a single unique ID. For example, let us recall L1, L2 and L3, and

Table I. Location table of KULOCS

Location label (PK)	Service	Actual location information
`kobe_univ`	gps01	`{latitude:35.4313, longitude:135.147, address:"1-1 Rokkodai Nada Kobe Japan"}`
`casher@ShopABC`	ips01	`ShopABC, (3.0, 4.5, 0.5)`
S101@kobe_univ	ips02	`KobeUniv.Lab.S101`

suppose that all of tktk in L1, Takatsuka in L2 and Object123 in L3 refer to the same person "hiroki". Then, the alias table contains an element: `{"id":"hiroki", "alias":{"L1":"tktk","L2":"Takatsuka", "L3":"Object123"}`. With this information, KULOCS converts the representative name hiroki into a real user ID when querying each of locating services. The integration of IDs can be also implemented with *common identity services* (e.g. OpenID[8]). However, it is beyond this paper's scope.

Based on the above design principle, KULOCS unifies L1, L2 and L3 as shown in Table II. Through KULOCS, the location data from any locating service are unified into the abstract location data with [when], [where] and [who].

2.3 Operation integration

We then propose KULOCS-API, which integrates heterogeneous operations of the existing locating services. Basically, KULOCS-API is an interface for querying KULOCS about a location of a mobile user (or object). The way of the query must be technology-neutral and independent of any specific locating services. Therefore, we again focus on the elements of [when], [where] and [who].

According to the possible combinations of the three elements, we derived six methods for KULOCS-API, as shown in Table III. For example, `where(time, id)` is for asking [where] based on known `time` (i.e. [when]) and `ID` (i.e. [who]). Thus, a user can invoke `where(NOW, B)` to know "Where is B (now)?". To achieve programmable interoperability, we publish KULOCS-API as a Web service, and deploy it in a cloud. For example, the method invocation `where(NOW, B)` can be performed in REST format `http://kulocs/where?time=NOW&id=B`.

Once the method of KULOCS-API is invoked, KULOCS internally *converts* the method invocation into an appropriate API call for each locating service (Figure 2). For the purpose of the method conversion, KULOCS manages the *service database*. Figure 3 shows the model diagram of KULOCS which indicates relations of three entities, data schemes and examples.

The service database has three entities: *service*, *api* and *param*. The *service* entity manages master information of all the underlying locating services. The information includes a name, an endpoint of the service, a type of the return value. In Figure 3, we can

Data ID	When/time	Where/location	Who/ID
L1	1434869412	`kobe_univ`	`hiroki`
L2	1435592713	`casher@ShopABC`	`hiroki`
L3	1435585774	`S101@kobe_univ`	`hiroki`

Table II. Data integration of L1, L2 and L3

Method	Description
`when(location, id)`	Returns the latest time when the object is in the location
`where(time, id)`	Returns the location where the object exists in the time
`who(time, location)`	Returns all objects that exist at the location in the time
`whenwhere(id)`	Returns a list of [time, location] where the given object exists
`whenwho(location)`	Returns a list of [time, id] that exist in the given location
`wherewho(time)`	Returns a list of [location, id] are located within the given time

Table III. List of methods in KULOCS-API

see that there are two locating services (LOCS4Geolocation, iBeaconLocator) registered. For each service, the *api* entity manages the mapping from the six methods of KULOCS-API to actual API in the service. In Figure 3, we can see that the `where()` method is mapped into `getLocation()` for gps01 (i.e. LOCS4Geolocation). The *param* entity manages the mapping and order of parameters within every method of KULOCS-API and the ones within the actual API call. For example, we can see, in Figure 3, that `time` and `ID` parameters of `where(time, id)` method are respectively passed to `time` and `user` parameters of `getLocation(user, time)` of gps01. Thus, the method can be converted.

Figure 4 shows a sequence diagram, where the user executes `where(NOW, B)` of KULOCS-API. In this scenario, KULOCS first finds a service gps01 from the service DB, and then identifies `getLocation()` API and its parameters' `user` and `time`. Next, KULOCS looks up the alias table to convert the ID of "B" into the local name "tktk" within gps01. Next, it invokes `getLocation()` of LOCS4Geolocation service with tktk and the current time, to locate tktk. Finally, the obtained location information is

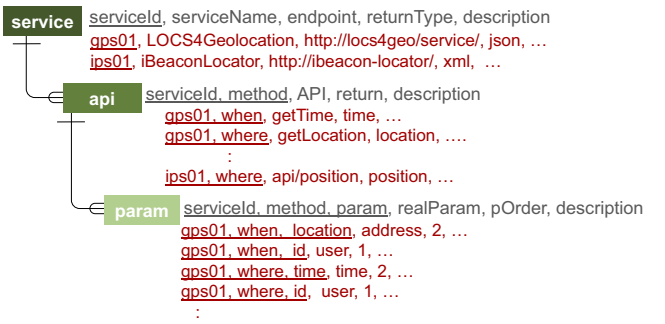

Figure 3.
Model diagram of KULOCS service database

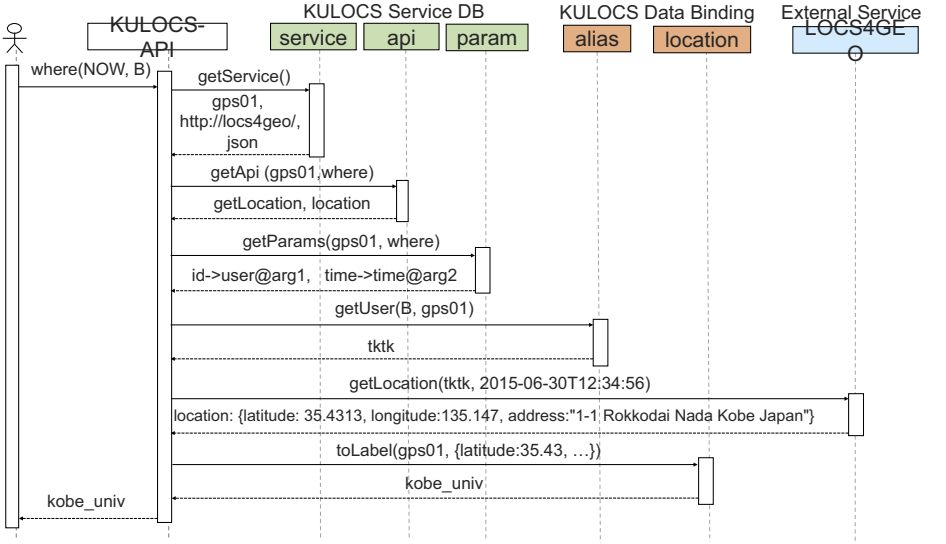

Figure 4.
Sequence diagram of KULOCS-API, in which where(NOW, B) is executed

converted into a location label with the location table. Finally, the label `kobe_univ` is returned to the user, as the answer of `where(NOW, B)`. Similarly, KULOCS can invoke any other locating service for `where(NOW, B)`. However, the sequence is omitted due to limited space.

3. System design and implementation

3.1 Detailed design

To implement KULOCS, we conduct an object-oriented design. Figure 5 shows the class diagram. We explain the detail of each class as follows.

KULOCSController. `KULOCSController` class works as a facade of all the underlying classes. It defines the six methods of KULOCS-API. Each method internally accesses the related databases and services as explained in Section 2.3, and returns an object of the corresponding *result class*. For instance, the `where()` method is executed as shown in Figure 4, and the result is returned by an object of Where. As mentioned in Section 2.3, `KULOCSController` is published as a Web service. Thus, every method can be executed by Web service protocols (REST and SOAP), so that client applications can use KULOCS from various kinds of platforms.

ThreeWs. ThreeWs class is an abstract class of the six result classes. It contains common information used in KULOCS-API, including parameters of a given query, error message and execution time. More specifically:

- *`message:`* an error message of API execution;
- *`timeQuery:`* a time parameter of the query;
- *`locationQuery:`* a location parameter of the query;
- *`idQuery:`* ID parameter of the query; and
- *`executionTime:`* an execution time of the API.

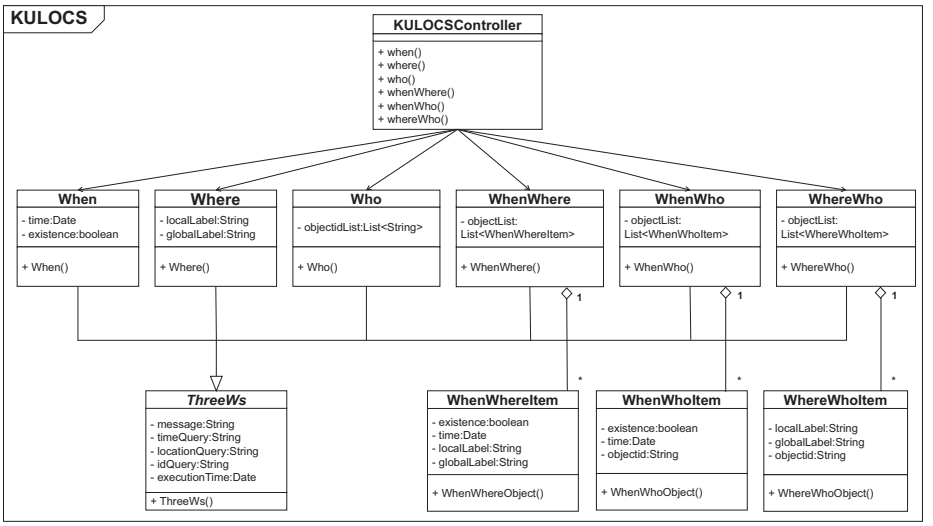

Figure 5.
Class diagram of KULOCS

When. When class is the result class of `when(location, id)`. As the answer of the query, the class contains the latest time (time), specifying when the object is in the given location. Also, it has `existence`, indicating whether the given object is there now. Clients of KULOCS-API typically ask if the target object is currently in the given location. The `existence` attribute helps such clients to save the effort for parsing `time`. In the current version, `existence` takes the true value if `time` is within 1 minute from now.

- `time`: the latest time when the object is in the location; and
- `existence`: a flag indicating the object is currently there.

Where. `Where` class is the result class of `where(time, id)`. As the answer of the query, the class contains the location where the object exist(ed) in the given time. The returned location is represented by `localLabel or globalLabel.`

- `localLabel`: an indoor location where the object exists in given time; and
- `globalLabel`: an outdoor location where the object exists in given time.

Who. Who class is the result class of `who(time, location)`. As the answer of the query, the class contains the list `(objectidList)` of IDs of all objects who exist(ed) in the given time in the designated location.

- `objectidList`: a List of IDs of all objects which exist(ed) in the given time and in the given location.

WhenWhere. `WhenWhere` class is the result class of `whenwhere(id)`. As the answer of the query, the class contains a list `(objectList)` of `WhenWhereItem`, representing a history of when and where the given object has been.

WhenWho. `WhenWho` class is the result class of `whenwho(location)`. As the answer of the query, the class contains a list `(objectList)` of `WhenWhoItem`, representing a history that when and who has existed in the given location.

WhereWho. `WhereWho` class is the result class of `wherewho(time)`. As the answer of the query, the class contains a list `(objectList)` of WhereWhoItem, representing a snapshot of the entire locating services of where and who exist(ed) in the given time.

3.2 Implementation

Based on the detailed design, we have implemented KULCOS. The total system comprised around 4,000 lines of code, and the development effort was three man-months. Technologies used for the implementation are as follows:

- *Language*: Java 1.7.0_85;
- *Web server*: Apache tomcat 7.0.39;
- *Web service framework*: Jersey 2.5.1;
- *Backend database*: MySQL 5.1.36; and
- *Server spec*: CentOS 6.4, dual-core CPU 2GHz, 4GB memory.

To show the practical feasibility of KULOCS, we have also implemented two locating services: *BLE Locating Service* and *GPS Locating Service*.

Bluetooth Low Energy Locating Service. BLE (Kohne and Sieck, 2014) is a short-range wireless communication technology, which can be used to detect the proximity of mobile objects. By deploying multiple BLE devices (called *beacons*) within indoor space, it is possible to implement an IPS based on the proximity. Our research group has been developing such a BLE-based IPS using BLE-equipped tablets (as mobile clients) and BLE hardware modules (as beacons). By wrapping the above IPS, we have developed a locating service, which we call *BLE Locating Service* in this paper. The technologies used for implementing the service are as follows:

- *Mobile client*: Google Nexus 7 (Android 5.0.2);
- *Data collector*: Android native application;
- *BLE beacons*: Aplix MyBeacon MB004 at-SR[9];
- *Language*: Java 1.7.0_85;
- *Web server*: Apache tomcat 7.0.39;
- *Web service framework*: Jersey 2.5.1;
- *Response format*: XML;
- *Backend database*: mySQL 5.1.36; and
- *Server spec*: CentOS 6.4, dual-core CPU 2GHz, 4GB memory.

The *BLE Locating Service* is integrated with KULOCS as one of the locating services.

GPS Locating Service. We have also implemented another locating service for outdoor space, using GPS sensors of a smart phone. We call this service *GPS Locating Service* in this paper. In the service, each mobile client (in outdoor space) periodically uploads the current location obtained by GPS to the server. The server provides the location data for authorized client applications via Web-API. The *GPS Locating Service* has been implemented with the following technologies:

- *Mobile client*: SHARP AQUOS PHONE SERIE SHL22 (Android 4.2.2);
- *Data collector*: Android native application;
- *Language*: Java 1.7.0_85;
- *Web server*: Apache tomcat 7.0.39;
- *Web service framework*: Jersey 2.5.1;
- *Response format*: JSON;
- *Backend database*: MongoDB 2.4.5; and
- *Server spec*: CentOS 6.4, dual-core CPU 2GHz, 4GB memory.

Compared to the *BLE Locating Service*, we intentionally used different technologies for response format and the backend database. This is to illustrate how KULOCS can accommodate the heterogeneity. The *GPS Locating Service* is also integrated with KULOCS as one of the locating services.

4. Developing application services with KULOCS

On top of KULOCS implemented in the previous section, we have developed two practical application services: *Umbrella Reminder Service* and *Stay Areas Visualization Service*.

4.1 Umbrella Reminder Service

The *Umbrella Reminder Service* prompts a user, who is leaving home, to take an umbrella when it is raining. In this service, KULOCS is used to evaluate the *location context* (Abowd *et al.*, 1991) that "a user is about to leave home". The context is defined by the fact that a user gets close to an entrance of a house, which is easily detected by KULOCS-API, e.g. `who(NOW, ENTRANCE@MYHOUSE)` .

To bind some actions to the location context, we used *RuCAS* (Takatsuka *et al.*, 2014), which was developed in our previous work. RuCAS is a framework that creates context-aware services using Web services. In RuCAS, every context-aware service is defined as an *ECA* (*Event–Condition–Action*) *rule* such that "when an event occurs, if a condition is satisfied, do designated actions".

Thus, the *Umbrella Reminder Service* has been implemented with RuCAS and KULOCS as follows:

- `Event`: A user is going to leave home (actually our laboratory). The context is defined as a situation that somebody is at the entrance, detected by KULOCS.
- *Condition*: It is raining outside. The context is defined as a fact that a weather forecast Web service indicates that it is rainy today.
- `Action`: Trigger a speech reminder "Do you have an umbrella?" using a Text-to-Speech Web service.

Figure 6 shows a screenshot of the user interface of RuCAS, which displays the page of a detailed *ECA* rule. The list in the left side of the page shows registered contexts and actions for select. The pane in the right side represents a created ECA rule, *Umbrella Reminder Service*.

Thus, the *Umbrella Reminder Service* implements a scenario that: when a user leave home, if the weather of today is rainy, the system speaks to alert "Do you have an umbrella?".

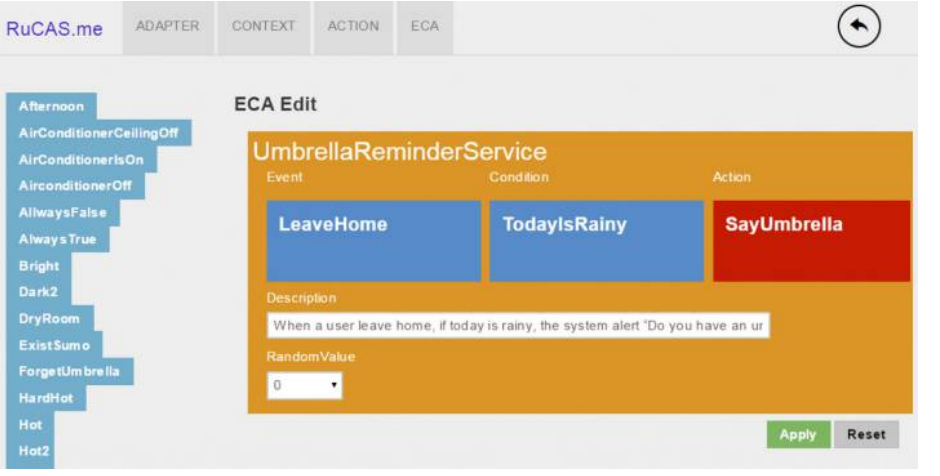

Figure 6.
Screenshot of *Umbrella Reminder Service*

4.2 Stay Areas Visualization Service

The *Stay Areas Visualization Service* is a Web service that displays the history of areas, where selected users have visited on the specified day. It is implemented by `whenwhere()` of KULOCS-API, JavaScript, HTML and CSS. Figure 7 shows a screenshot of the service. The vertical axis indicates the hours and the horizontal axis indicates the users. We can see in the screenshot that on July 31, 2015, the user tktk went to his desk of his laboratory at 8:00 a.m. and worked until 12:00 p.m. Then, tktk went a meal in the cafeteria at 12:00 p.m. and worked until 7:00 p.m. After eating dinner, tktk went home. Similarly, another user horihori went to the izakaya, where he works part-time, from 6:00 pm, and the user takatori worked until late after the dinner.

Note that the service can display the log of various locations seamlessly, regardless that the locations are inside or outside. This is the great advantage of KULOCS that can horizontally integrate heterogeneous locating services.

5. Experimental evaluation

5.1 Application development with or without KULOCS

To demonstrate the practical effectiveness, we conducted an experiment, where we investigate two cases of application development. The one is with KULOCS, and the other is with the conventional manual integration of locating services. Intuitively, the experiment is to see the difference between Figures 1 and 2.

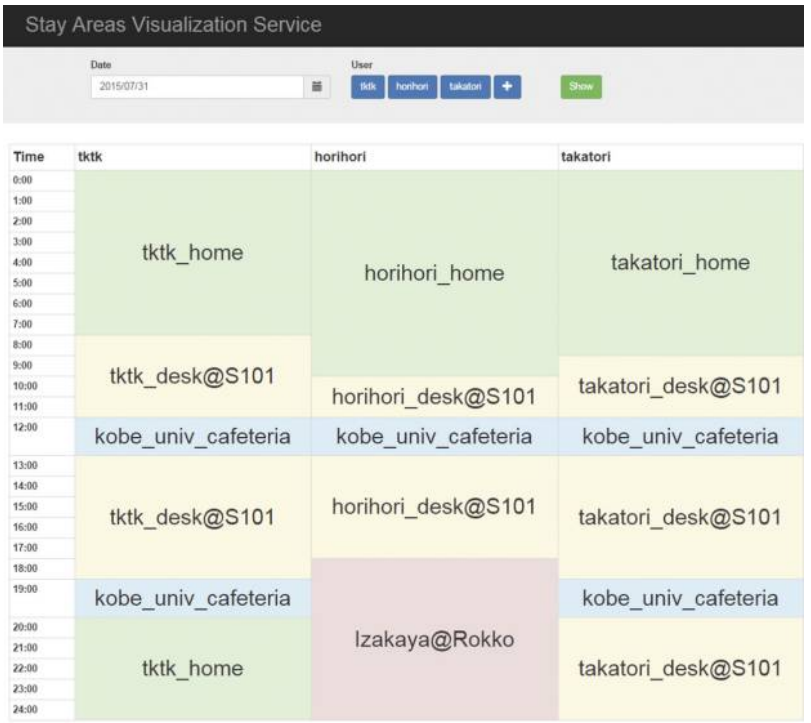

Figure 7.
Screenshot of *Stay Areas Visualization Service*

In the experiment, we implement two versions of a Web application, either of which returns the current location of a given user. The implementation language used for the both versions is Node.js. The one version is implemented with the developed KULOCS, whereas the other version directly uses the API of the *BLE Locating Service* and the *GPS Locating Service* (see Section 3.2).

Table IV shows the lines of code and the response time of the applications. The response time is the average time of ten executions. The two versions were executed in the same condition that:

- the user queries the location of a user tktk [e.g. `where(now, tktk)] of KULOCS-API`; and
- tktk is in `kobe_univ` and is located by the GPS Locating Service.

We can see in Table IV that using KULOCS reduces about 34 per cent of the code from the conventional application. One may think that it is not a drastic reduction. This is, however, justified by the fact that there were only two locating services in the experiment (i.e. the *BLE Locating Service* and the *GPS Locating Service*). Thus, the conventional integration did not become much complicated. If the number of locating services becomes larger, the developer has to integrate heterogeneous API and data format by himself, which requires more time and effort. In that case, the benefit of KULOCS becomes much more significant.

In Table IV, we can see that KULOCS imposes small performance overhead compared to the conventional application. However, according to the investigation, we found that most of the response time is spent in the underlying locating services, and that the overhead is so small that it cannot be a serious issue of the application execution. Thus, we can see that by using KULOCS, a developer can implement location-based applications efficiently without a performance problem.

5.2 Performance evaluation of KULOCS-API

As shown in Table III, KULOCS-API consists of six different methods. These six methods can be used for different purposes. However, in some use cases, one can implement the same feature with different methods. For instance, suppose that a developer wants to check a context "tktk is in Kobe University now" in the application. Then, the developer can use any of the six methods to implement it, which yields a design choice. Now our interest here is which method should be chosen for the better implementation.

Table V compares the six methods, where each method is used to evaluate "tktk is in Kobe University now". The second column represents parameters necessary for each method to locate tktk at Kobe University. The third column represents the total response time for executing the corresponding method. The fourth and fifth columns represent the response time spent in KULOCS and the locating services, respectively. Each value of the response time is the average value for ten measurements.

Table IV.
Comparison of two versions with KULOCS and with the conventional integration

Used service	Lines of code	Response time (ms)
Individual locating services	53	273.9
KULOCS	35	289.2

In Table V, we can see that there is not much difference in response time among `when()`, `where()` and `who()`, as well as among `whenwhere()`, `whenwho()` and `wherewho()`. However, there is a big performance gap between the two groups. One reason of the gap is that `whenwhere()` and `wherewho()` scan all the locating services to extract the history of location data, which is quite time-consuming. Moreover, `whenwhere()`, (`whenwho()` and `wherewho()` as well return a list of objects, which imposes expensive data parsing on KULOCS.

Thus, when a developer has a design choice, the best way is to try to use `when()`, `where()` or `who()` as much as possible. In the case of checking "tktk is in Kobe University now", using `when()` [or `where()`] is the good choice in the perspectives of performance and intuition.

5.3 Applicability to practical services

To show further potential of KULOCS, here we try to develop ideas of other practical services enabled by KULOCS:

- *Time card service*: This service provides a capability of a time card, which automatically manages how long a user has been staying at a certain place. The service can be implemented with `when()` of KULOCS-API. Typical use cases include the attendance management of a company, car parking and unified management of rental space by the hour (e.g. karaoke rooms).
- *Seamless tracking service*: This service displays user's current location on a map (e.g. Google Map) seamlessly, regardless of the location being indoor or outdoor. This service can be implemented with `where()` of KULOCS-API. A user no longer needs to switch among different maps for different locating services.
- *Attendance checking service*: This service allows a user to check who and how many people are attending in a certain place. The service can be implemented with `who()` of KULOCS-API. Typical use cases include counting participants in an event and checking attendance in a college class.
- *Guestbook service*: This service automatically generates a guestbook recording of who came when at a certain place. The service can be implemented with `whenwho()` of KULOCS-API. Typical use cases include counting visitors to a touristic place (e.g. shrine and temple) and checking guests in a ceremony (e.g. wedding).
- *Travel companion reviewing service*: This service allows a user to recall who the user traveled with. The service can be implemented with `wherewho()` of

API	Parameters	Total RT (ms)	RT of KULOCS (ms)	RT of LS (ms)
`when()`	location=kobe_univ&id=tktk	34.5	10.4	24.1
`where()`	time=now&id=tktk	24.4	11.6	12.8
`who()`	time=now&location=kobe_univ	45.4	31.1	14.3
`whenwhere()`	id=tktk	198.5	117.2	81.3
`whenwho()`	location=kobe univ	201.7	11.0	190.7
`wherewho()`	time=now	179.2	91.1	88.1

Table V. Response time of KULOCS-API

KULOCS-API. Reviewing the travel log with the companion information may motivate the user to do better future travels. For instance, a user may think: "I found that I did not travel much with my family recently. So I will spare more time with my family for the next holiday".

6. Discussion

6.1 Related work

Ficco *et al.* (2014) proposed a hybrid location system, which combines wireless fingerprinting technologies for indoor positioning together with GPS-based positioning for outdoor localization. As a user moves to different places, the system autonomously switches to the best available positioning method supported by the mobile device and the surrounding environment. This study mainly focuses on the switching mechanism in the mobile clients. However, it does not cover how to integrate the existing locating services and location data. Thus, the significant difference is that they try to integrate different positioning systems within the client side, which heavily relies on the capability of the mobile device. On the other hand, we try to integrate them within the server side, which does not rely on any capability of clients.

Ahn and Nah (2010) proposed a Web service framework based on service-oriented architecture, called *LOCA* (*LOcation-based Context-Aware Web services*) *framework*. LOCA discovers available Web services based on client location information and preference. Thus, a client can dynamically find, integrate and consume Web service available in the current location. The difference from our approach is that LOCA provides a location-based service discovery, while KULOCS provides a location query portal for any location-based services. In this sense, LOCA can integrate KULOCS to manage wider locations efficiently.

Christensen *et al.* (2015) proposed *Searchlight Graph* (*SLG*) and *Searchlight Continuous Query Processing Framework* (*CQPF*). SLG is a directed graph representing the topology of multiple locations (indoor and outdoor), where each location is associated with mobile objects. Using CQPF with an SQL-like language, a user can query the past, current or future location of a mobile object within the SLG. Their approach of representing location as a point on the graph is similar to our thought of using a location label in KULOCS. Compared to KULOCS, Searchlight allows more detailed location queries with topology information (range, area, etc.). However, the topology is limited within the SLG, and interoperability among different locating services is not well considered. On the other hand, KULOCS does not currently support any topology, as it is abstracted during the data integration. We consider it a tradeoff between a framework compromising different location models and a framework imposing special constraints for the location model. Indeed, it is interesting to consider how to manage topological information within KULOCS, which will be left for our future work.

6.2 Stakeholders around KULOCS

When putting KULOCS into practice, we have to clarify roles and positions of the surrounding *stakeholders*. We here consider three types of stakeholders: *provider*, *developer* and *end-user*. Figure 8 shows relationships among the three stakeholders and KULOCS applications/services.

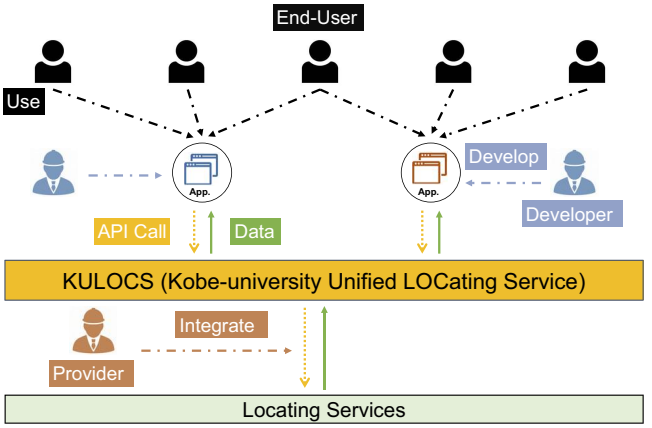

Figure 8.
Relationship among KULOCS, provider, users, developers and applications

The provider is a stakeholder who actually provides KULOCS. Basically, KULOCS is provided as a *singleton* service deployed on a cloud. The operation and maintenance of KULOCS is the main role of the provider. In addition, the provider is responsible to integrate the existing locating services. For every given locating service, the provider connects the service to KULOCS by the proposed data and the operation integration methods. More specifically, the provider registers new entries in the service DB, as well as creates data binding definition of the new service. The provider may implement an adapter, which converts a response format of Web-API into a format of KULOCS, if necessary.

The developer is a stakeholder who implements location-based applications and services using KULOCS. Thus, the developer is the direct user of KULOCS. Using KULOCS-API, the developer can implement location-based application efficiently, without knowing the details of individual locating services. Thus, the main goal of KULOCS is to support the developers.

The end-user is a stakeholder who uses the location-based applications developed by the developers. For each application, we basically assume that each location label is registered by the developer of the application, and is shared with the end-users. However, depending on the application, the developer should provide a *location registration feature*, with which the end-users by themselves can register and share own location labels.

To support efficient management of location labels, we are currently developing a service, which provides API to help register of the location labels. To maintain the uniqueness of the location label, we encourage using the location label as a primary key in the location database. By doing so, KULOCS can easily reject a duplicated location label during the location registration.

6.3 Authenticating existing locating services

KULOCS integrates the existing locating services, some of which may require user's login to access the location information. Therefore, we need to consider how to authenticate these services. To cope with the challenge, we consider using OAuth 2.0[10], which provides an authentication method for Web services and resources.

Figure 9 shows a sequence diagram of OAuth 2.0 with a location-based application using KULOCS. In the sequence, the application first requests to execute KULOCS-API (without an *access token*). Then, KULOCS redirects the request to an authentication server of the existing locating service. The authentication server requires a user of the application to login to authorize the resource access. After the successful authorization, the authentication server redirects the application with an access token. Again, the application executes the KULOCS-API with the access token. Using the token, KULOCS executes the API of the locating service to access the location information. The resource server of the locating service checks if the access token is valid, and returns the requested location information to KULOCS. The information is converted via KULOCS data integration, and finally returned to the application.

Thus, OAuth 2.0 can be used to authenticate the existing services, simply and effectively.

6.4 Security and privacy issues

The security and privacy issues are also a challenging topic. KULOCS and the location-based applications (e.g. the ones proposed in Section 5.3) make full use of location information gathered from users. The location information of a user is personal information, which might be abused by other users (e.g. theft, stalking). Therefore, for every location-based application, it is important to carefully consider *operation policies*, which strictly define how and by whom the location information is used for what.

Indeed, this is not a specific issue with KULOCS only, but also is seen in many other social networking service (SNS)- and location-based services. Because KULOCS works as an abstraction layer of the underlying locating services, the operation policies of KULOCS must be derived from the operation policies of the underlying services.

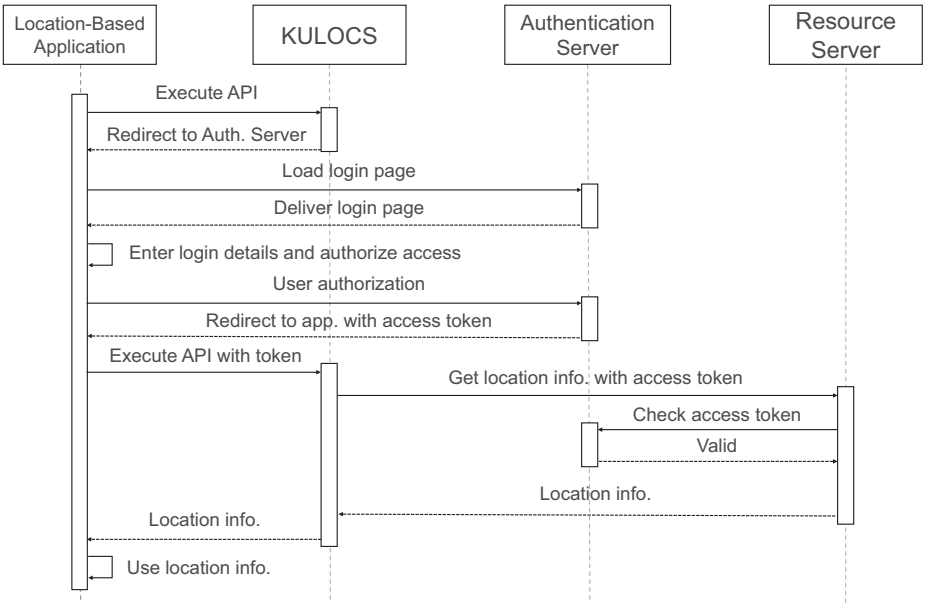

Figure 9.
Sequence diagram of OAuth 2.0 with location-based application using KULOCS

Preferably, KULOCS should be able to provide API that facilitates *user opt-in*, by which a user declares the scope and range of the location information to be published. Considering reasonable policies for security and privacy will be left for our future work.

7. Conclusion

In this paper, we have proposed a unified locating service, called KULOCS. To integrate the existing heterogeneous locating services, KULOCS was designed to achieve data integration and operation integration. Based on technology-neutral elements [when], [where] and [who], we proposed a method of the data integration with Unix time, the location label and the alias table. For the operation integration, we propose KULOCS-API, with the six methods derived from the combination of the three elements.

We have also implemented KULOCS and underlying locating services (*BLE Locating Service* and *GPS Locating Service*). On top of the implementation, we developed two application services to demonstrate the practical feasibility. In the experimental evaluation, we conducted application development with and without KULOCS. The result shows that KULOCS reduces the effort of application development, especially when the number of locating services becomes large. We also discussed the performance of KULOCS-API and the applicability to more practical services.

Finally, we summarize our future work. A challenging topic is to consider how to cope with the security and privacy issues when integrating multiple locating services. We are also interested in how to preserve topological information in the data integration of KULOCS.

Notes

1. Available at: www.skyhookwireless.com/.
2. Available at: www.swarmapp.com/.
3. Available at: www.glympse.com/.
4. Available at: https://developers.google.com/maps/.
5. Available at: https://pathsha.re/.
6. Available at: twww.apple.com/ios/whats-new/family-sharing/.
7. Available at: www.indooratlas.com/.
8. Available at: http://openid.net/specs/openid-connect-core-1_0.html.
9. Available at: www.aplix.co.jp/?page_id=10721.
10. Available at: http://oauth.net/2/.

References

Abowd, G.D., Dey, A.K., Brown, P.J., Davies, N., Smith, M. and Steggles, P. (1991), "Towards a better understanding of context and context-awareness", *Handheld and Ubiquitous Computing*, Springer, New York, NY, pp. 304-307.

Ahn, C. and Nah, Y. (2010), "Design of location-based web service framework for context-aware applications in ubiquitous environments", *2010 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, *IEEE, Newport Beach, CA*, pp. 426-433.

Christensen, K.F., Christiansen, L., Pedersen, T.B. and Pihl, J. (2015), "Searchlight: context-aware predictive Continuous Querying of moving objects in symbolic space", *IEEE 31st International Conference on Data Engineering (ICDE)*, IEEE, Seoul, pp. 687-698.

Ficco, M., Palmieri, F. and Castiglione, A. (2014), "Hybrid indoor and outdoor location services for new generation mobile terminals", *Personal and Ubiquitous Computing*, Vol. 18 No. 2, pp. 271-285.

Kohne, M. and Sieck, J. (2014), "Location-based services with iBeacon technology", *2nd International Conference on Artificial Intelligence, Modelling and Simulation*, *IEEE, Madrid*, pp. 315-321.

Manandhar, D. and Torimoto, H. (2011), "Opening up indoors: Japan's indoor messaging system, IMES", available at: http://gpsworld.com/wirelessindoor-positioningopening-up-indoors-11603/ (accessed: 19 January 2016).

Pratama, A.R. and Hidayat, R. (2012), "Smartphone-based pedestrian dead reckoning as an indoor positioning system", *International Conference on System Engineering and Technology*, *IEEE, Bandung*, pp. 1-6.

Takatsuka, H., Saiki, S., Matsumoto, S. and Nakamura, M. (2014), "Design and implementation of rule-based framework for context-aware services with web services", *The 16th International Conference on Information Integration and Web-based Applications & Services*, *ACM*, Hanoi, pp. 233-242.

Takatsuka, H., Tokunaga, S., Saiki, S., Matsumoto, S. and Nakamura, M. (2015), "Integrating heterogeneous locating services for efficient development of location-based services", *The 17th International Conference on Information Integration and Web-based Applications & Services*, *ACM*, Belgium, pp. 430-439.

Ting, S., Kwok, S.K., Tsang, A.H. and Ho, G.T. (2011), "The study on using passive RFID tags for indoor positioning", *International Journal of Engineering Business Management*, Vol. 3 No. 1, pp. 9-15.

Vermesan, O., Friess, P., Guillemin, P., Gusmeroli, S., Sundmaeker, H., Bassi, A., Jubert, I.S., Mazura, M., Harrison, M. and Eisenhauer, M. (2011), "Internet of things strategic research roadmap", *Internet of Things: Global Technological and Societal Trends*, available at: www.internet-of-things-research.eu/pdf/IoT_Cluster_Strategic_Research_Agenda_2011.pdf (accessed 19 January 2016).

Corresponding author
Hiroki Takatsuka can be contacted at: tktk@ws.cs.kobe-u.ac.jp