On Integrating Heterogeneous Locating Services

Hiroki Takatsuka, Sachio Saiki, Shinsuke Matsumoto, and Masahide Nakamura

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

Abstract. This paper presents a unified locating service, KULOCS, which horizontally integrates the existing heterogeneous locating services. Focusing on technology-independent elements [when], [where] and [who] in querying locations of objects, KULOCS integrates data and operations of the existing services. In the data integration, we propose a method where the time representation, the locations, the namespace of user are consolidated by Unix time, the location labels and the alias table, respectively. We then propose KULOCS-API that integrates operations by all possible combinations of [when], [where] and [who]. Since KULOCS works as a seamless façade to the underlying locating services, clients can consume location information easily and efficiently, without knowing concrete services actually locating target objects. Also, we examine feasibility of two practical value-added services with KULOCS.

Keywords: locating service, indoor positioning system, location information, Web services, location-aware service

1 Introduction

Smart coupling of IoT, positioning systems and cloud technologies enables an extensible infrastructure to acquire and manage *locations* of users and objects. Nowadays, every smartphone is equipped with GPS. Also, various GPS modules for IoT appear on the market (e.g., [4]). The emerging *indoor positioning systems (IPS)* can locate users even inside buildings or underground, where GPS cannot cover. The enabling technologies of IPS include Wi-Fi [6], Bluetooth beacons [10], RFID [12], IMES [11]. Gathering such indoor/outdoor location information in the cloud would create a 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 via API, without knowing implementation details of underlying positioning systems. We call such a cloud service *locating service* in this paper. In fact, several practical services come onto market recently. They include Swarm [7], Glympse [1], Pathshare [5], and IndoorAtlas [2]. Although features and operation policies vary from one 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 API for developers.

Basically, there is no compatibility among different locating services and API, since they are individually developed and operated. Each service is tightly coupled with the underlying positioning system. For example, Glympse assumes to

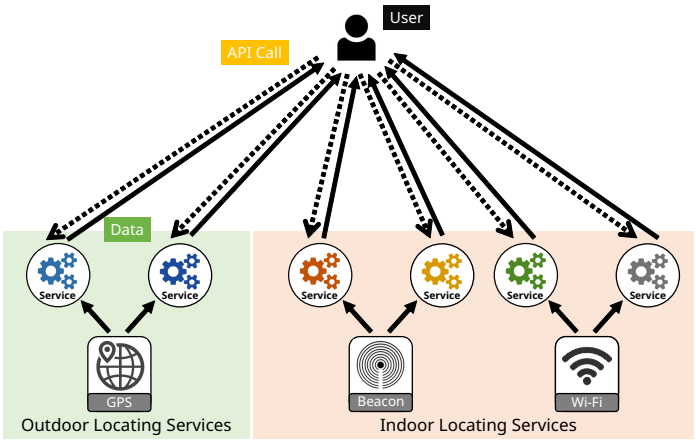

Fig. 1. Conventional architecture to integrate locating services

use GPS information collected by smartphones, while IndoorAtlas use a magnetic field to locate the position inside a building. Thus, Glympse cannot directly use the data of IndoorAtlas, and vice versa. In order 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.

Fig. 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 queries the location of another mobile user B. The user B is either inside or outside of any building, and is supposed to be located by a certain locating service. Then, "where-are-you?" executed by A invokes different API for all possible locating services, in order to find B. Although A's the query "Where is B?" is essentially simple, the application has to know how to query and interpret the location for individual locating services, respectively. This makes the application quite complex, low-performance, and non-scalable.

To cope with the problem, we propose in this paper 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 that different representation of time, heterogeneous locations and different namespace of a user 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 user of KULOCS queries every location by a label, whereas KULOCS internally converts the label to specific representation for individual locating services.

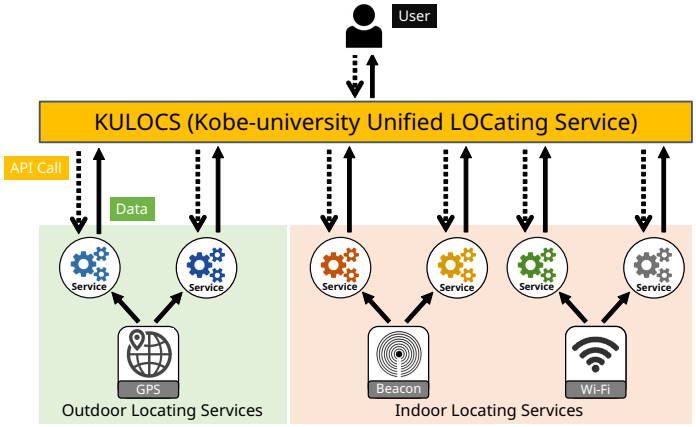

Fig. 2. Architecture of KULOCS

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 Web service, so that applications on various platforms 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 needs not to know how B is located by which service. Thus, the application can consume location information quite easily and efficiently.

To show the practical feasibility of KULOCS, we examine two practical value-added services with KULOCS. One example is Seamless Locating Service, which allows a user to locate a mobile object in either indoor or outdoor space. The another example is Personalized Location-Aware Service. A user can create and customize own location-aware services by associating a location and an action. We discuss how these two services can be implemented by KULOCS.

2 KULOCS (Kobe-University Unified Locating Service)

2.1 Overview

We propose *KULOCS (Kobe-university Unified LOCating Service)* in this section. Fig. 2 shows its architecture. KULOCS works as a *façade* 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. Since KULOCS is an abstract layer that integrates heterogeneous locating services, we have to achieve the followings:

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: Also, 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 necessary for any services to locate an object:

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

KULOCS is designed to accept *generic* location queries based on possible combinations of the above three elements. KULOCS then translates the generic query to service-specific queries for individual services.

2.2 Data Integration

We here describe how to integrate location data of heterogeneous locating services. To help understanding, 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 is 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 respectively use different time representation (and time zone).

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

As for [where], there are many ways and different granularity levels to represent a location. The GPS coordinate looks 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

Table 1. Location table of KULOCS

Loc. label (PK)	Service	Actual Location info.
`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

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 1. 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], since every locating service has 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 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 [3]). However, it is beyond this paper.

Based on the above design principle, KULOCS unifies L1, L2 and L3 as shown in Table 2. Through KULOCS, the location data from any locating service is unified into the abstract location data with [when], [where] and [who].

Table 2. Data integration of L1, L2 and L3

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

2.3 Operation Integration

We then propose KULOCS-API, which integrates heterogeneous operations of the existing locating services. Basically, KULOCS-API is the 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 3. 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

Table 3. List of methods in KULOCS-API

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 who exist in the locaion 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.

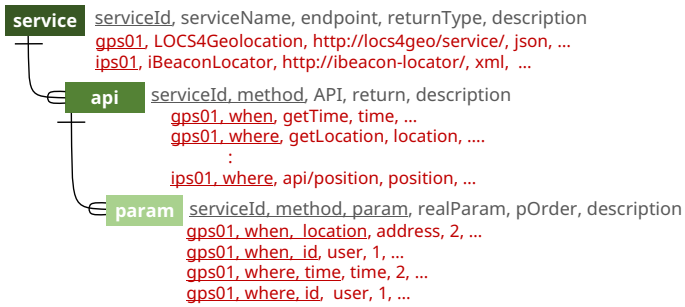

Fig. 3. ER diagram of KULOCS service database

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 a method of KULOCS-API is invoked, KULOCS internally *converts* the method invocation into an appropriate API call for each locating service (see Fig. 2). For the purpose of the method conversion, KULOCS manages the *service database*. Fig. 3 shows the ER diagram of the service database.

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 Fig. 3, we can 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 Fig. 3, we can see that `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 Fig. 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.

Fig. 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 converted into a location label with

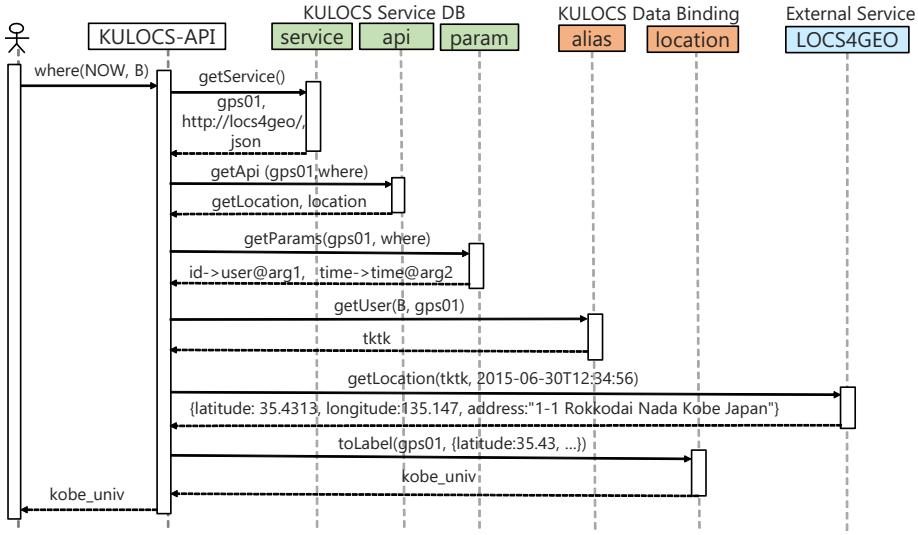

Fig. 4. Sequence diagram of KULOCS-API, in which `where(NOW, B)` is executed

the location table. Finally, the label `kobe_univ` is returned to the user, as the answer of `where(NOW, B)`. Similarly, KULOCS can invoke other locating services for `where(NOW, B)`. However, the sequences are omitted due to limited space.

2.4 Implementation

We are currently implementing KULOCS as a Java Web service. The implementation details and evaluation are left for our future publications. We are also interested in pragmatic issues, including security and privacy policies in location-based services. These are also planed for our future research.

3 Creating Value-added Services with KULOCS

The proposed KULOCS unifies heterogeneous locating services, seamlessly. We examine here two practical services enabled by KULOCS.

Seamless Locating Service: For a given ID of a registered mobile user, this service locates the user regardless the user is in indoor or outdoor place. Although this service uses all possible locating services, there is no need to take care of proprietary communications, thanking KULOCS.

Personalized Location-Aware Service: KULOCS allows a user to easily evaluate *location context*, which returns true when the user come to a pre-registered location. Binding a location context and a certain action implements a *location-aware service*. If a system allows the user to register favorite locations and binding rules, it implements *personalized location-aware services*. For example, when the user gets close to home, the lights are turned on. When the user sits on a sofa in a living room, a music is automatically played back.

4 Related Work

Ficco et al. [9] 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 available positioning method supported by the mobile device and the surrounding environment. This study mainly focuses on the switching mechanism in the mobile clients. Thus, the difference is that they try to integrate different positioning systems within client side, which relies on the capability of the device. On the other hand, we try to integrate them within the server side, which does not rely on any capability of clients.

Ahn et al. [8] proposed a web service framework based on service-oriented architecture, called *LOCA* . It discovers best-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 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 exploit KULOCS for more extensive location management.

Acknowledgements: 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)].

References

1. Glympse. https://www.glympse.com/
2. IndoorAtlas. https://www.indooratlas.com/
3. OpenID. http://openid.net/specs/openid-connect-core-1_0.html
4. OriginGPS. http://www.origingps.com/
5. Pathshare. https://pathsha.re/
6. Skyhook. http://www.skyhookwireless.com/
7. Swarm by foursquare. https://www.swarmapp.com/
8. Ahn, C., Nah, Y.: Design of location-based web service framework for context-aware applications in ubiquitous environments. In: 2010 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing. pp. 426–433
9. Ficco, M., Palmieri, F., Castiglione, A.: Hybrid indoor and outdoor location services for new generation mobile terminals. Personal and Ubiquitous Computing 18(2), 271–285 (2014)
10. Kohne, M., Sieck, J.: Location-based services with ibeacon technology. In: 2014 2nd International Conference on Artificial Intelligence, Modelling and Simulation. pp. 315–321
11. Manandhar, D., Torimoto, H.: Opening up indoors: Japan's indoor messaging system, IMES (2011), http://gpsworld.com/wirelessindoor-positioningopening-up-indoors-11603/
12. Ting, S., Kwok, S.K., Tsang, A.H., Ho, G.T.: The study on using passive RFID tags for indoor positioning. International journal of engineering business management 3, 9–15 (2011)