Integrating Heterogeneous Locating Services for Efficient Development of Location-Based Services

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

ABSTRACT

This paper proposes a unified locating service, KULOCS, which horizontally integrates the heterogeneous locating services. 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, we propose a method where the time representation, the locations, the namespace are consolidated by Unix time, the location labels and the alias table, respectively. Based on possible combinations of the three elements, we then derive API for the operation integration.

In this paper, we also implement KULOCS as a Java Web service and integrate two locating services: GPS-based outdoor locating service and BLE-based indoor locating service. On top of the implementation, we develop application services: Umbrella Reminder Service and Stay Areas Visualization Service. Experimental evaluation shows the practical feasibility by comparing cases with or without KULOCS. Since KULOCS works as a seamless façade to the underlying locating services, the users and applications consume location information easily and efficiently, without knowing concrete services actually locating target objects.

Categories and Subject Descriptors

H.3.4 [**Systems and Software**]: Distributed systems; H.3.5 [**Online Information Services**]: Web-based services; D.2.12 [**Interoperability**]: Data mapping

General Terms

Design, Experimentation

Keywords

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

1. INTRODUCTION

Smart combination of IoT, 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 GPS. Also, various GPS modules for IoT appear on the market (e.g., [7][11]). 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 [9], Bluetooth beacons [15], RFID [20], Pedestrian Dead Reckoning (PDR) [17], IMES [16]. 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 *locating service* in this paper. In fact, several practical services come onto market recently. They include Swarm [10], Glympse [3], Google Maps APIs [4], Pathshare [8], Apple Family Sharing [2] and IndoorAtlas [5]. 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, since they are individually developed and operated. Each service is tightly coupled with the underlying positioning system. For example, Glympse assumes to 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.

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-performance, and non-scalable.

To cope with the problem, in this paper, we propose a uni-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
iiWAS2015, 11-13 December, 2015, Brussels, Belgium.
Copyright 2015 ACM 978-1-4503-3491-4/15/12 ...$15.00.

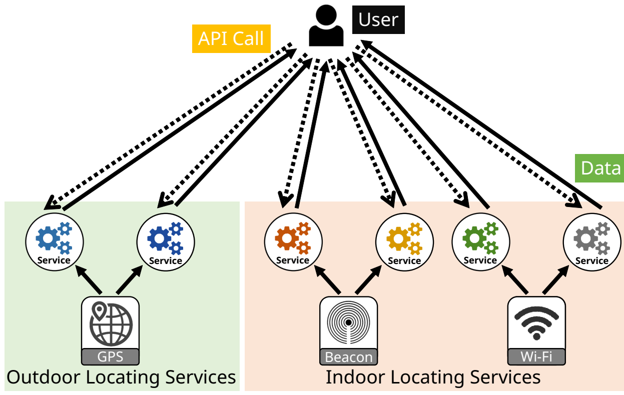

Figure 1: Conventional architecture to integrate locating services

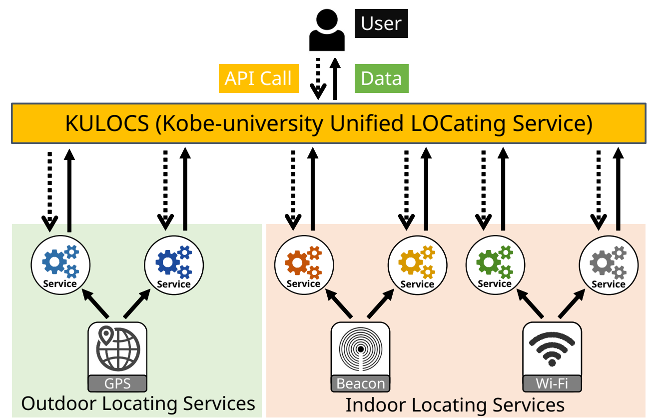

Figure 2: Architecture of KULOCS

fied 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 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 user of KULOCS queries every location by a label, whereas KULOCS internally converts the label to 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 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 needs not to know how B is located by which service. Thus, the application can consume location information quite easily and efficiently.

In this paper, we also design and implement the proposed KULOCS as a Java Web service. The current version supports to integrate 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 that "when a user leaves home", 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 that 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 concept of KULOCS has been published in [19]. Changes were made on this conference paper, most significantly the addition of design and implementation of KULOCS, the application services, and the experimental evaluation (Sections 3, 4 and 5). 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 basic principles of *KULOCS (Kobe-university Unified LOCating Service)*. Figure 2 shows the 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 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:

Table 1: 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

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`

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 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.

- **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.

Note that other interrogatives like how, what and why are not included since 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.

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 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 location data, we consider the elements [when], [where], [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 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 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*

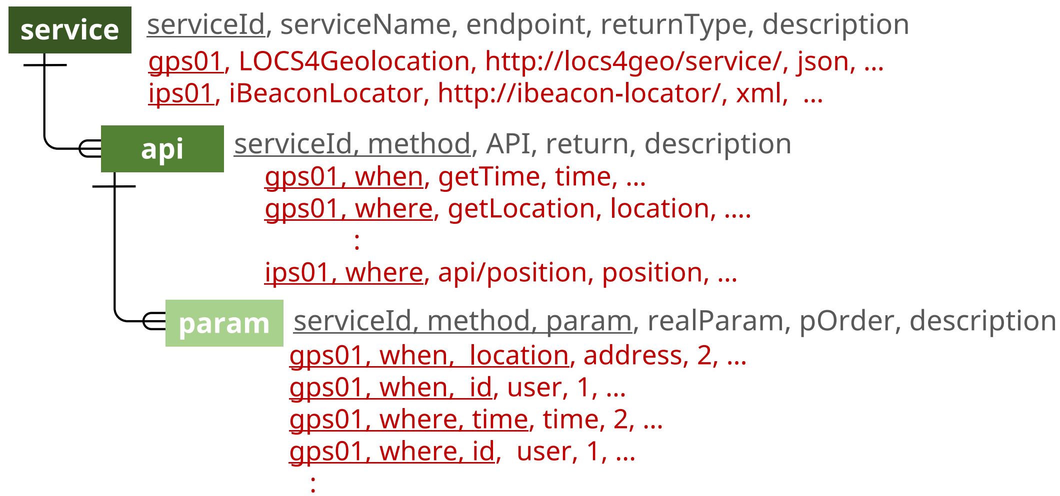

Figure 3: Model diagram of KULOCS service database

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 [6]). 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].

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 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 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 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 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 `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 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

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

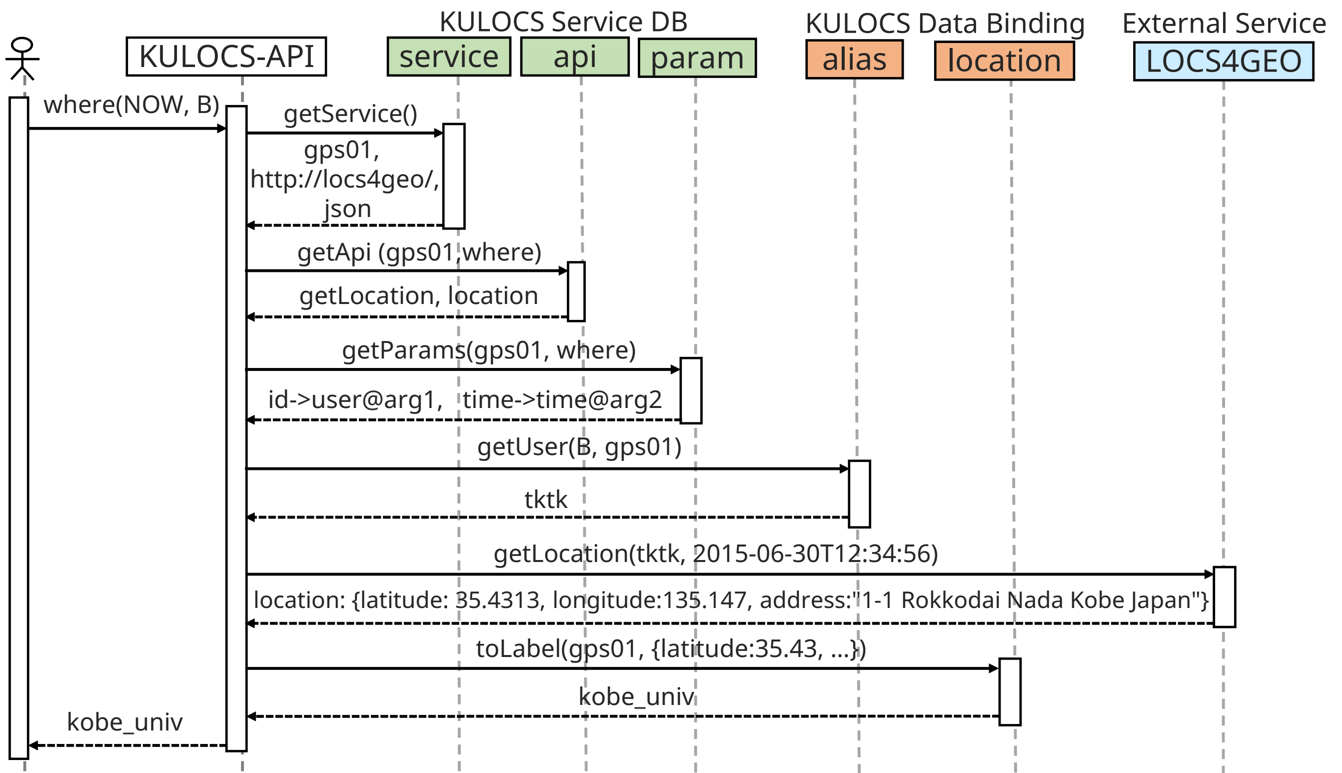

Figure 4: Sequence diagram of KULOCS-API, in which `where(NOW, B)` is executed

KULOCSController

`KULOCSController` class works as a façade 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, `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.
- `executionTime`: An execution time of the API.

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 or not 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 one minute from now.

- `time`: The latest time when the object is in the location.
- `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.
- `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 ID's of all objects who exist(ed) in the given time in the designated location.

- `objectidList`: A List of ID's of all objects which exist(ed) in the given time and in the given location.

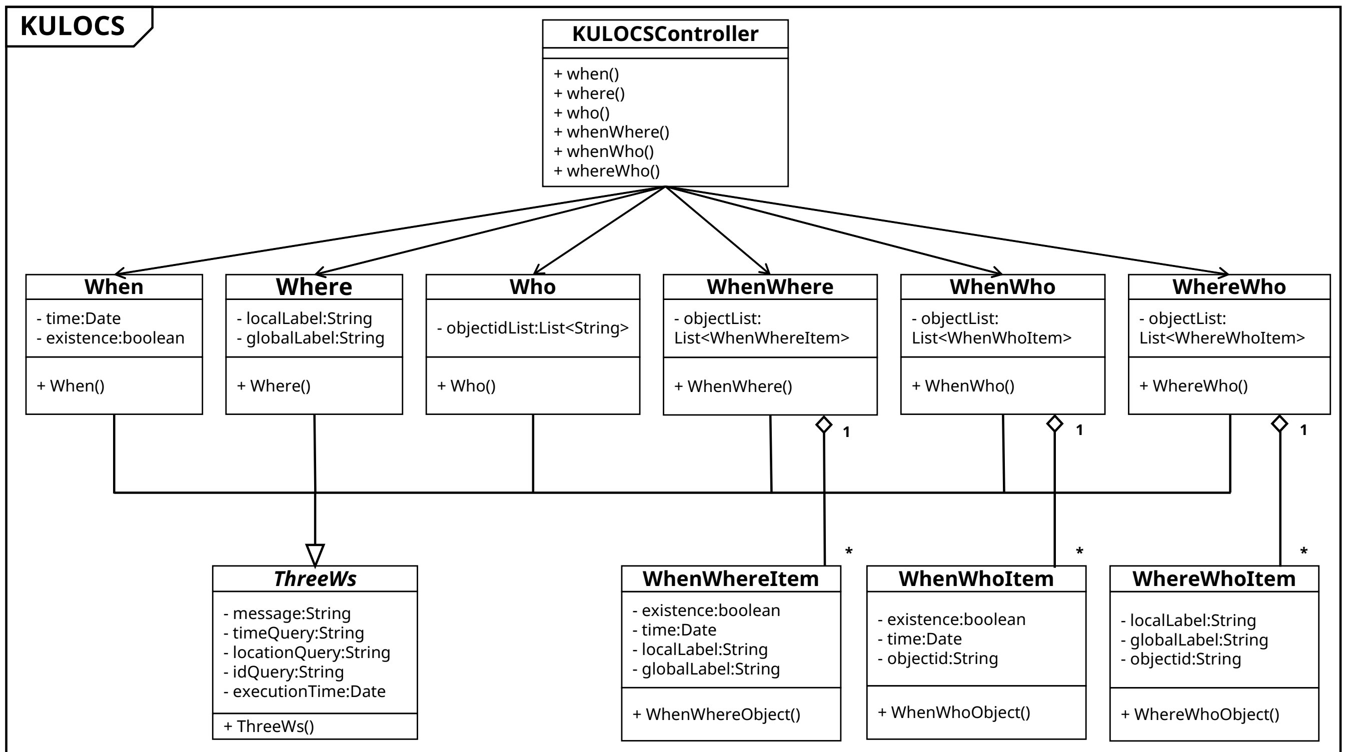

Figure 5: Class diagram of KULOCS

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 that 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 whole locating services that 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 of 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
- **Server spec:** CentOS 6.4, Dual-Core CPU 2GHz, 4GB memory

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

BLE Locating Service

Bluetooth Low Energy (BLE) [15] 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 [1]
- **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
- **Server spec:** CentOS 6.4, Dual-Core CPU 2GHz, 4GB memory

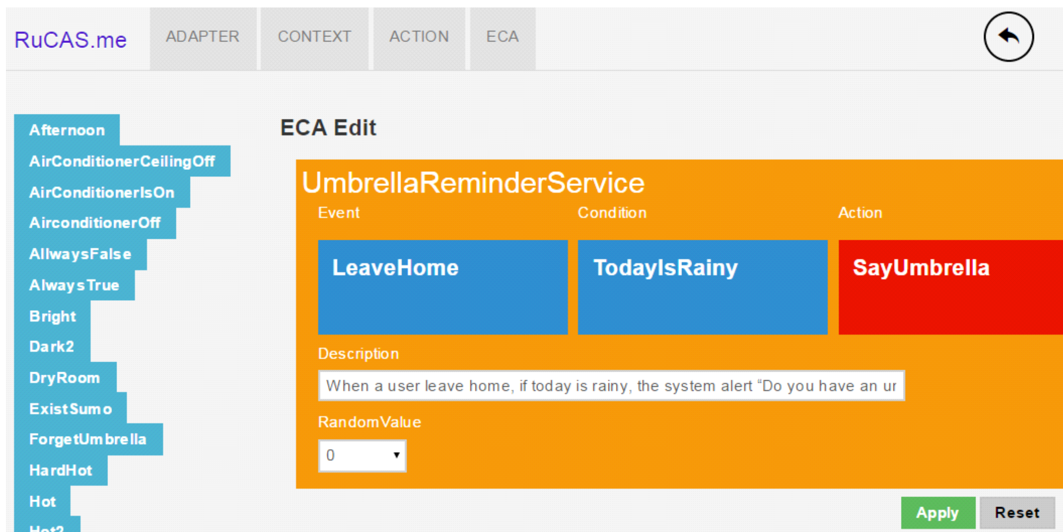

Figure 6: Screenshot of Umbrella Reminder Service

The BLE Locating Service is integrated with KULOCS as one of 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, every 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
- **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* that "a user is about to leave home". The context is defined by the fact a user gets close to an entrance of a house, which is easily detected by KULOCS-API, e.g., `who(NOW, ENTRANCE@MYHOUSE)`.

In order to bind some actions to the location context, we used *RuCAS* [18], 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 rain 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, where 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?".

4.2 Stay Areas Visualization Service

Table 4: 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

Table 5: Response time of KULOCS-API

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

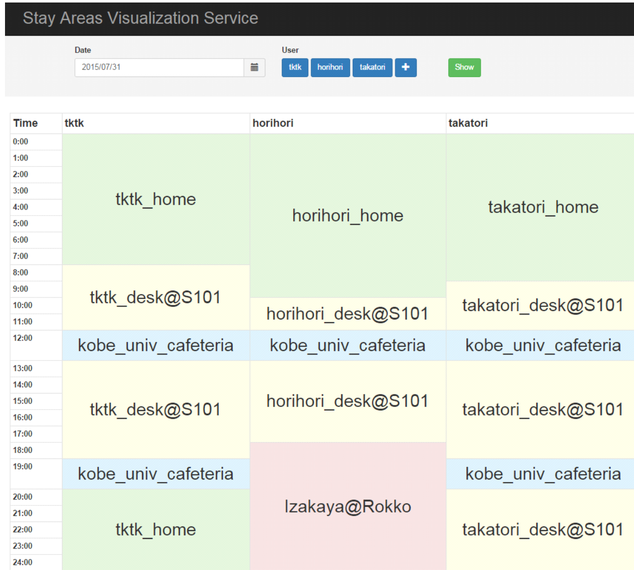

Figure 7: Screenshot of Stay Areas Visualization Service

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 31st, 2015, the user tktk went to his desk of his laboratory at 8 o'clock and worked until 12 o'clock. Then, tktk went a meal in the cafeteria at 12 o'clock and worked until 19 o'clock. After eating dinner, tktk went to the home. Similarly, another user horihori went to the izakaya, where is a place of his part-time job, from 18 o'clock, 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 here conduct 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 Figure 1 and Figure 2.

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 4 shows the lines of code and the response time of the applications. The response time is the average time of 10 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`.
- tktk is in `kobe_univ` and is located by the GPS Locating Service.

We can see in Table 4 that using KULOCS reduces about 34% 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 complicated very much. 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 4, 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 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 3, 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 5 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 in 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.

In Table 5, we can see that there is no much difference in response time among `when(), where(), who()`, as well as among `whenwhere(), whenwho(), 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) returns 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 the location is 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 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 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.".

These practical services make full use of location information gathered from users. Therefore, it is important to carefully consider *operation policies* of the services, which addresses the security and privacy of the users. Indeed, this is not a specific issue with KULOCS only, but also is seen in many other SNS and location-based services. Considering reasonable policies for security and privacy will be left for our future work.

6. RELATED WORK

Ficco et al. [14] 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 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 et al. [12] 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. [13] 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 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 are 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 trade-off 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.

7. CONCLUSION

In this paper, we have proposed a unified locating service, called KULOCS. In order 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.

8. ACKNOWLEDGMENTS

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)].

9. REFERENCES

[1] Aplix MyBeacon MB004 At-SR. http://www.aplix.co.jp/?page_id=10721. Accessed: 2015-09-01.

[2] Family Sharing. http://www.apple.com/ios/whats-new/family-sharing/. Accessed: 2015-09-01.

[3] Glympse. https://www.glympse.com/. Accessed: 2015-09-01.

[4] Google Maps APIs. https://developers.google.com/maps/. Accessed: 2015-09-01.

[5] IndoorAtlas. https://www.indooratlas.com/. Accessed: 2015-09-01.

[6] OpenID. http://openid.net/specs/openid-connect-core-1_0.html. Accessed: 2015-09-01.

[7] OriginGPS. http://www.origingps.com/. Accessed: 2015-09-01.

[8] Pathshare. https://pathsha.re/. Accessed: 2015-09-01.

[9] Skyhook. http://www.skyhookwireless.com/. Accessed: 2015-09-01.

[10] Swarm by foursquare. https://www.swarmapp.com/. Accessed: 2015-09-01.

[11] Tiny GPS module. http://kilohertz.io/portfolio/tiny-gps-module/. Accessed: 2015-09-01.

[12] C. Ahn and Y. Nah. 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*, pages 426–433, June 2010.

[13] K. Christensen, L. Linnerup Christiansen, T. Pedersen, and J. Pihl. Searchlight: Context-aware predictive continuous querying of moving objects in symbolic space. In *2015 IEEE 31st International Conference on Data Engineering*, pages 687–698, April 2015.

[14] M. Ficco, F. Palmieri, and A. Castiglione. Hybrid indoor and outdoor location services for new generation mobile terminals. *Personal and Ubiquitous Computing*, 18(2):271–285, 2014.

[15] M. Kohne and J. Sieck. Location-based services with ibeacon technology. In *2014 2nd International Conference on Artificial Intelligence, Modelling and Simulation*, pages 315–321, November 2014.

[16] D. Manandhar and H. Torimoto. Opening up indoors: Japan's indoor messaging system, IMES, 2011. http://gpsworld.com/wirelessindoor-positioningopening-up-indoors-11603/ Accessed: 2015-09-01.

[17] A. R. Pratama, R. Hidayat, et al. Smartphone-based pedestrian dead reckoning as an indoor positioning system. In *2012 International Conference on System Engineering and Technology*, pages 1–6. IEEE, September 2012.

[18] H. Takatsuka, S. Saiki, S. Matsumoto, and M. Nakamura. Design and implementation of rule-based framework for context-aware services with web services. In *The 16th International Conference on Information Integration and Web-based Applications & Services*, pages 233–242, December 2014.

[19] H. Takatsuka, S. Saiki, S. Matsumoto, and M. Nakamura. On integrating heterogeneous locating services. In *2nd EAI International Conference on IoT as a Service*, October 2015. (to appear).

[20] S. Ting, S. K. Kwok, A. H. Tsang, and G. T. Ho. The study on using passive RFID tags for indoor positioning. *International journal of engineering business management*, 3:9–15, 2011.