Implementation and Evaluation of Cloud-based Integration Framework for Indoor Location

Long Niu, Sachio Saiki, Shinsuke Matsumoto, Masahide Nakamura
Graduate School of System Informatics, Kobe University
longniu@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

The emerging indoor positioning systems (IPS) enable indoor location-aware applications (InL-App) within indoor space where GPS cannot reach. In most conventional systems, however, IPS and InL-App are tightly coupled, where one system cannot reuse location data or operation of other systems. This fact yields expensive development cost and effort of InL-App. To cope with the problem, this paper propose a cloud-based integration framework, called CIF4InL. With a common data model, CIF4InL integrates indoor location data obtained from heterogeneous IPS. It then provides application-neutral API for various InL-Apps. To evaluate the practical feasibility, we integrate two different IPS (RedPin and BluePin) using CIF4InL, where the applications transparently access the indoor locations gathered by two different IPS. Since CIF4InL allows the loose coupling between IPS and InL-Apps, it significantly improves reusability of indoor location information and operation.

Categories and Subject Descriptors

H.2.1 [**Logical Design**]: Data models; H.2.8 [**Database Applications**]: Spatial databases and GIS; H.3.4 [**Systems and Software**]: Distributed systems; H.3.5 [**Online Information Services**]: Web-based services, Data sharing

Keywords

Indoor positioning framework, location information, data modeling, location-aware service, API, indoor location query service

1. INTRODUCTION

With the rapid development of wireless and sensor technologies, *Indoor Positioning System* (IPS, for short) is attracting great attention in recent years. IPS is a system that locates and tracks objects within indoor space (inside buildings, underground, and so on) where Global Positioning System (GPS) does not work well. Compared to outdoor space, within indoor space one can easily deploy extra infrastructure for the positioning system. There are many research and development of IPS in the last decade. Enabling technologies of IPS include sound or ultrasound [4], image analysis [18], RFID [14], Wi-Fi [16], other radio-based approaches [11]. Although the goals of these IPS are the same, they are different in many aspects, such as system topology, measuring principles, positioning algorithms, location information (data model).

By using IPS, various Indoor Location-aware Applications (we call indoor-location application, or InL-App for short) can be implemented. An InL-App performs appropriate actions autonomously, according to indoor location of users or dynamic/static objects. Typical use cases include searching goods or equipment in a hospital, locating of firemen in a building on fire, detecting the location of police dogs trained to find explosives in a building, and finding tagged maintenance tools and equipment scattered all over a plant.

When we implement an InL-App with an IPS, it is necessary to determine, within the InL-App, how to represent and manage indoor-location data provided by IPS. Most conventional systems individually define and manage the indoor location information, considering the purpose of the InL-App and characteristics of the IPS used. Such a proprietary representation and management method has an advantage of optimal performance. However, it causes "tight coupling" between InL-App and the underlying IPS, where indoor-location data and operations cannot be reused among different InL-App. Thus, the proprietary method makes the implementation of InL-App complicated, and increases development cost and effort.

As the first step to cope with problem, we have previously proposed a common data model for indoor location, call DM4InL [15]. The DM4InL defines a common data schema for representing indoor location information of various objects (people, appliance, room, spot, and so on) without depending on any specific IPS or InL-App.

Following the previous achievement of DM4InL, the main concern of this paper is how to construct a framework, where various InL-Apps can easily share and consume indoor location information gathered by various IPS. For this, we present the Cloud-based Integration Framework for Indoor Location (CIF4InL, for short) in this paper. Using DM4InL, the proposed CIF4InL integrates indoor location data obtained from existing heterogeneous IPS, and provides common operation as a service for various InL-Apps. Concerning this, we have to tackle two challenges. The first challenge is data integration, i.e., how to convert indoor location

iiWAS '15, December 11-13, 2015, Brussels, Belgium

DOI: http://dx.doi.org/10.1145/2837185.2837220

data produced by heterogeneous IPS into DM4InL. The next challenge is operation integration, i.e., how to implement comprehensive location-based queries retrieving data from DM4InL, to be shared by various InL-Apps.

To overcome those challenges, we design CIF4InL based on following three components: (1) InL-Adapter, (2)InL-Database, (3)InL-Query. InL-Adapter adapts the proprietary indoor location data to common data model DM4InL. The InL-Adapter converts the uploaded location data into DM4InL. InL-Database is a large-scale shared database that manages the translated data. InL-Query provides application-neutral API for various InL-Apps to query the indoor location information. Operations for these components are published as cloud services, and thus they are loosely coupled by service-oriented architecture.

To evaluate practical feasibility, we apply the proposed framework to integrate two different IPS. The first IPS is RedPin [1], which uses Wi-Fi fingerprints to locate mobile devices. The second IPS is BluePin, which uses Bluetooth beacons for detecting proximity of the devices. The proposed CIF4InL integrates the two different IPS, so that applications can transparently uses indoor location information gathered by both systems. It is unnecessary for the applications to manage the difference of RedPin and BluePin. Since CIF4InL allows the loose coupling between IPS and InL-Apps, it improves reusability and interoperability of indoor location information and operation. Thus, it is promising to reduce development cost and effort of InL-App, significantly.

2. PRELIMINARIES

2.1 Indoor Positioning System (IPS)

The indoor positioning system (IPS) generally refers to a system that estimates position of subject or object within indoor space. The primary progress in IPS has been made during the last ten years. Therefore, there is no de-facto standard for the IPS yet, compared to GPS. In general, IPS can divided into several categories:

- **Vision Based Indoor Localization:** Visual information can be collected and practiced for indoor navigation in many literature (e.g., [21][13]). However, the image based localization will consume more computing resource (analyzing the image) and power.
- **Wireless Based Indoor Localization:** Unlike light, wireless wave can get through doors and walls and provide ubiquitous coverage of a building. The development with the existing wireless technologies (e.g., Wi-Fi, Bluetooth) is relatively easy, and the microwaves does not obstruct human activities in the building. Moreover, the power and computing resource consumption also significantly less than vision based indoor localization. Most current work in indoor localization use this way.
- **Other Methods:** There also many other ways for indoor localization. They include ultrasound [4], acoustic background fingerprint [17], accelerometer [8], and campus by adopting a dead-reckon method [9].

Among recent literatures, the wireless based indoor localization methods take up most proportion of them. According to mathematical techniques used, they can categorize them into the following three groups:

- **Proximity:** This method assumes that if a user enter within the range of a known station, then location of the user is approximated to the point of the station. We are currently developing an IPS, called *BluePin*, using Bluetooth Beacon technology. On detecting proximity of a user, BluePin produces symbolic location data. The following data symbolic location data. The following data L1 represents that user `niu` get closed an entrance of room S101.

  ```
  L1:{personId:niu, locationId:22, locationName:
  S101 Entrance, LastUpdate: 2015/07/27 11:23:45
  JST}
  ```

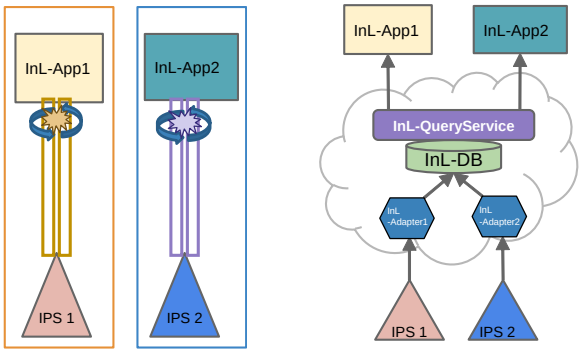
- **Triangulation:** This method uses geometric knowledge to obtain the user location. The location is determined by either the distance to the fixed known measurement points, or the received signal angles.
- **Fingerprint:** The fingerprint means the characteristic of feature of signals. The method assumes that each position in the area has a unique fingerprint. Relying on prior knowledge associating a fingerprint with a position, the current location of a user is obtained. For example, Redpin [1] is an open-source IPS which uses Wi-Fi fingerprint for zone-based positioning. The following data L2 is produced by RedPin, representing that a user is in location 45 of a room S103, pointed on (345, 567) on a map KU-System-1F:

  ```
  L2:{locationId:45, mapName:KU-System-1F, map
  Xcord:346, mapYcord:567, symbolicId:S103,
  macAddress:'08:60:6e:32:b6:0b'}
  ```

2.2 Indoor Location Application (InL-App)

In this paper, *Indoor Location Application (InL-App)* refer to any location-aware service or application that performs appropriate actions according to indoor location information. To help understanding, let us introduce the following examples:

- **SmartShop:** This service pushes coupons or loyalty program of a shop to a smartphone when a user approaches the shop. The user's location is estimated by BluePin, where a static beacon station is installed at the entrance of the shop. When a user 1 gets closed to the station, the user's smart phone upload the location data `{personId:1, locationId:7, locationName:shop1, lastUpdate:2015/07/27 11:23:45 JST}` to a server. Since SmartShop knows that user 1 in the shop, it pushes shop coupons to the user's smart phone.
- **LocEyes:** This service visualizes locations of all staff working in an institute. We assume that every staff has a smartphone with RedPin, and that the smartphone uploads the current indoor location every 10 seconds. An instance of the location is `{locationId:45, mapName:KU-System-1F, mapXcord:346, mapYcord:567, symbolicId:S103, macAddress:'08:60:6e:32:b6:0b'}`. According to the data, the server visualizes the latest location of every staff on the map KU-System-1F.



(a) conventional InL-App (b) proposed InL-App

Figure 1: Two different architectures of InL-App

It is easily to understand that there is no compatibility between SmartShop (with Bluepin) and LocEyes (with Redpin). Indeed, they are individually developed and operated, considering the service objectives and the underlying IPS. Figure1(a) shows the implementation architecture of these InL-Apps. We can see that each In-App is tightly coupled with an IPS, and that indoor location data and program are managed independently within each system. Therefore, one system cannot share or reuse the data and operation of another system. As a result, each InL-App has to be developed from scratch, which causes expensive cost and effort.

2.3 Previous Work: DM4InL

In our previous work [15], we have proposed DM4InL (Data Model for Indoor Locations). It defines a common data schema for representing indoor location information without depending on any specific IPS or InL-App. DM4InL consists of three models: location model, building model and object model. The location model represents spatial data in vector format and defines a global position for each building. The building model represents geographic elements, such as entrance, routes or room. The object model represents various mobile objects, such as human, robot or vehicle.

According to Yuan et al. [20], every spatial object must have theme, space and time attributes, to represent what, where and when, respectively. Hence, every object in the object model has an indoor-location point (in the location model), each of which is explained by spatial elements of a building (in the building model).

The original concept of DM4InL was published in a workshop paper [15]. Changes were made on this paper most significantly the addition of ObjectLocationLog, which stores the history of time-series indoor locations for every object. We will give some detailed descriptions of DM4InL in Section 3.3.

2.4 Long-Term Goal and Scope of Paper

As mentioned in Section 2.2, the conventional architecture lacks compatibility and reusability, due to the tightly coupling between InL-App and IPS. Therefore, our research goal is to establish an application platform as shown in Figure 1(b), which achieves loose coupling between InL-Apps and the underlying IPS. This paper focuses on implementing a framework that horizontally integrates the existing IPS and provides common operation as a service shared by various InL-Apps. To implement such a common framework, we have to tackle **two challenges**: *data integration* and *operation integration*. The data integration considers how to convert indoor location data produced by various IPS into the one conforming to a common data model. Following the previous work, we investigate how to convert the proprietary indoor location data into DM4InL. On the other hand, the operation integration deals with implement comprehensive queries retrieving application-neutral location data from DM4InL.

3. CIF4INL: CLOUD-BASED INTEGRATION FRAMEWORK FOR INDOOR LOCATION

3.1 Architecture

To overcome those challenges mentioned in Section 2.4, we propose CIF4InL *(Cloud-based Integration Framework for Indoor Location)*. CIF4InL works as an abstract layer between InL-App and IPS. This layer first integrates indoor location data gathered by heterogeneous IPS, and then provides application-neutral API for various InL-App, by which InL-App can access to different IPS transparently.

Figure 1(b) shows its architecture. The CIF4InL consists of three components: InL-Adapter, InL-Database, InL-Query. Features of each component are describe below:

- **InL-Adapter (Indoor Location Adapter Service)** This is a Web service that adapts the proprietary indoor location data to the common data model DM4InL. When a client uploads proprietary indoor location data via Web-API, InL-Adapter converts the data into the one in DM4InL, and inserts the converted data in a database (InL-Database, see below). Since different IPS create location data in different format, we need to implement a dedicated adapter for every IPS.
- **InL-Database (Indoor Location Database)** This is a large-scale shared database that manages the indoor location data provided by InL-Adapter. Every record of indoor location data complies with DM4InL, and is stored with time-stamp to keep the history.
- **InL-Query (Indoor Location Query Service)** This is a Web service for querying indoor location data stored in InL-Database. It provides application-neutral API for various InL-Apps to query indoor location of any object. InL-Query provides two types of API: fundamental API and composite API.

The whole CIF4InL itself is deployed as cloud service, where the above components are loosely coupled by service-oriented architecture (SOA).

3.2 Approach Overview

In order to manage the data integration and the operation integration, CIF4InL is designed specifically as follows:

- **Data Integration:** Heterogeneous indoor location data are managed in a single schema of DM4InL. The conversion from proprietary data format into DM4InL is conducted by individual InL-Adapter. By doing this, it is unnecessary to modify the existing IPS. A client just uploads the location data via Web-API of designated InL-Adapter, where all the tasks for the data conversion and storing are delegated to CIF4InL. The detail of InL-Adapter will be described in Section 4.
- **Operation Integration:** Heterogeneous operations for the existing IPS are consolidated by InL-Query, with which every InL-App can retrieve indoor location data in DM4InL. Each application does not need to know technical details of the underlying IPS. As will be shown in Section 5, InL-Query provides fundamental API and composite API.

3.3 Data Schema of DM4InL

Before going into the details, we briefly review data schema of DM4InL, since it is essential for CIF4InL. The DM4InL aims to prescribe a common data schema, independent of implementation of IPS or the usage of InL-App. It represents location of every indoor object with three kinds of models: location, building and object.

- **Location Model:** It represents any location in a building by a relative position (3-dimensional offset) from the base coordinates of the building. Using the coordinate, we construct four geometric primitives: local point, local line, local polygon and local space. It also defines global position in [longitude, latitude, altitude] and an angle formed by its X axis and the north direction.
- **Building Model:** It defines every building with theme attributes and global position. It also defines geographic elements in each building such as partitions, routes and spots. Each spot (route or partition) is located by a local point (a local line or a local space, respectively) in the location model. It also identifies every building with a reference point represented by a global position.
- **Object Model:** It defines various objects in building, such as people, appliance, furniture, and object location log stores location and time information of objects. Each object location log refers to a local point in the location model to represent its position.

Figure2 shows an ER diagram of DM4InL, representing relationships among the three models. The diagram follows the notation defined in [19]. A square represents an entity. A relationship may be defined between a pair of entities, where

- (+——$\in$) represents a parent-child relationship,
- (+——$\cdots$) represents a reference relationship,
- (+——$\circ$+) represents a sub-type relationship

Every indoor location (i.e., LP, LLN or LSP) is associated with a single building, whereas every building involves more than one indoor locations. A building (B) is located by a global position (GPos). A geographic element in a building (i.e., spot, route or partition) refers to a location entity (LP, LLN or LSP, respectively). An object location log refers to a local point and an object entity. The full description of DM4InL can be found in [15].

4. INL-ADAPTER FOR DATA INTEGRATION

4.1 Overview

To achieve the data integration of heterogeneous IPS, the proposed CIF4InL implements InL-Adapter *(Indoor Location Adapter Service)*. InL-Adapter is a Web service that adapts the proprietary indoor location data to DM4InL.

As mentioned in Section 3.2, clients of each type of IPS first uploads the proprietary location data to InL-Adapter, and then the InL-Adapter converts the data into the one with DM4InL.

To implement this, we have to address two issues: *topology adaptation* and *data conversion*. The topology adaptation considers the structure of how to upload the measured data to InL-Adapter, which will be described in Section 4.2. The data conversion considers how the InL-Adapter converts the uploaded data into DM4InL format, which will be described in Section 4.3.

4.2 Topology Adaptation

As a first step, we need to slightly modify the existing IPS, so that the measured indoor location data are uploaded to an InL-Adapter. For this modification, we have to consider the *system topology* of the IPS. However, the topology varies from one IPS to another. Therefore, we propose different adaptation patterns for different topology.

According to Liu et al. [10], there are four different system topology for IPS: *remote-positioning, self-positioning, indirect remote-positioning*, and *indirect self-positioning*.

Figure 3 shows the four topology. In the figure, a triangle represents a static device or station deployed in the infrastructure. A circle represents a mobile device to be located. A rectangle represents a server. A hexagon represents an InL-Adapter, to which we newly adapt the existing IPS. The labels "M", "R" and "T" represent roles of measuring unit, signal receiver, and signal transmitter, respectively.

Figure 3 (1) shows the remote-positioning topology, where the remote server locates the mobile device. The static stations receive the signal transmitted from the mobile device, and forward the signal to the server. The server then computes the location of the mobile device. An IPS with presence sensors in [7] belongs to this topology. In the remote-positioning topology, all the location data are managed in the server. Therefore, we modify the server so as to upload the measured data to an InL-Adapter.

Figure 3 (2) shows the self-positioning topology, where the mobile device itself measures the location. This mobile device receives signals from infrastructure, and computes the current location from the signals. IMES [12] and GPS belong to this topology. In the self-positioning topology, all

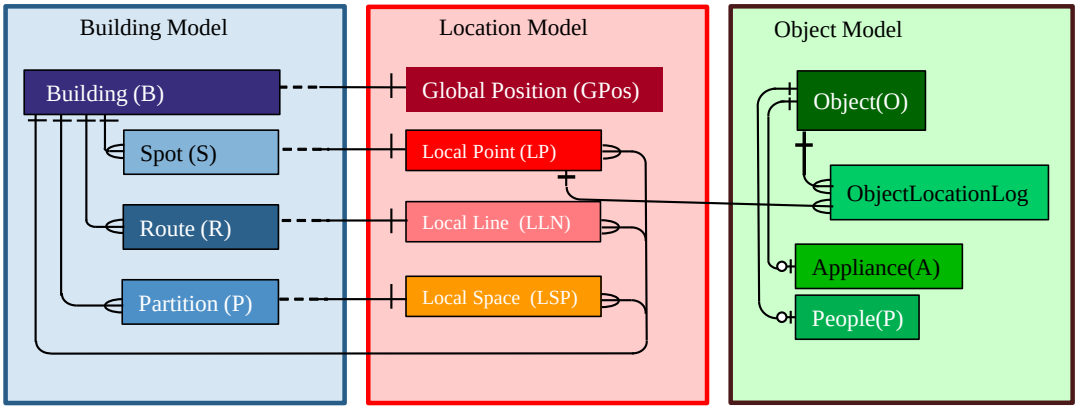


Figure 2: DM4InL as a composition of 3 models

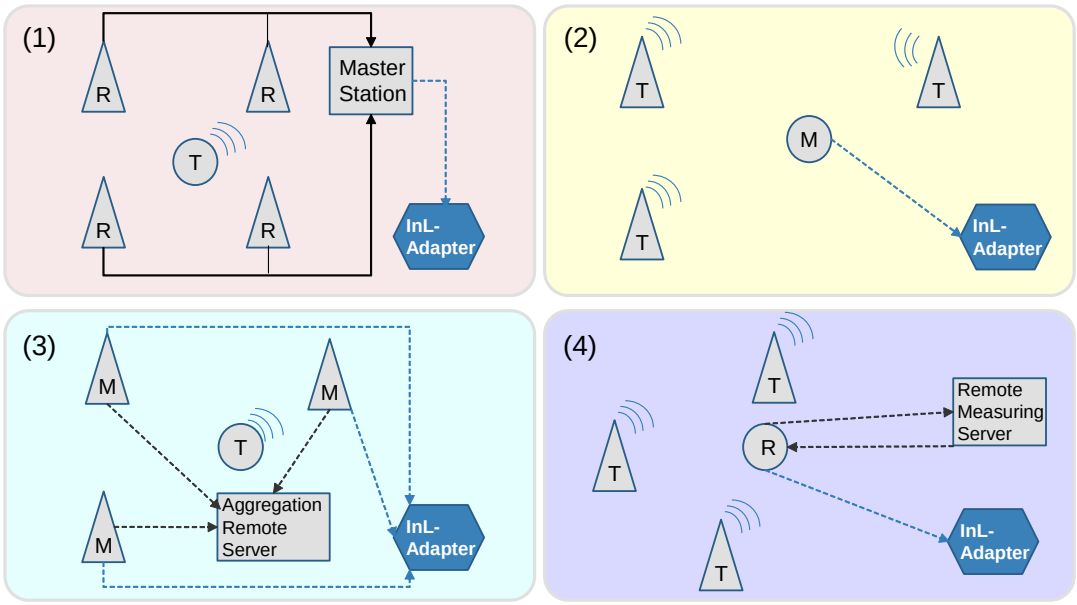


Figure 3: Four different IPS topologies and adaptation patterns

the location data are managed by the mobile device. Therefore, we modify the mobile device so as to upload measured location data to an InL-Adapter.

Figure 3 (3) shows the indirect remote-positioning topology, where the mobile device is located indirectly by the remote server. To cover wide area or multiple buildings, several stations with the measuring capability collaborate to send the location data to the aggregation server. In the indirect remote-positioning topology, the location data are managed by either the measuring stations or the aggregation server. Considering the fact that the aggregation server is often complex and is implemented with un-modifiable patent products, we choose to modify the stations to upload the data to an InL-Adapter.

Figure 3 (4) shows the indirect self-positioning topology, where the mobile device indirectly obtains its location via the remote server. The mobile device first receives signals from the infrastructure, and then forwards the signals to the remote server. The server computes the current location, and returns the location data to the mobile device. With the development of IoT and cloud technologies, this type of IPS are gaining popularity. RedPin and BluePin introduced in Section 2.1 belong to this topology. In the indirect self-positioning topology, the location data are held by either the mobile device or the measuring server. Considering the complexity and workload of the measuring server, we choose to modify the mobile device to upload the returned location data to an InL-Adapter.

4.3 Data Conversion

The second step is to consider how the InL-Adapter converts the uploaded location data into DM4InL. In general, every IPS defines its own location data format relying on the specific infrastructure and/or positioning algorithm. It is, therefore, impossible to enumerate data converters for all possible IPS in this paper. Instead, we present a *template* of how an InL-Adapter should convert the proprietary data into DM4InL. Then the template is validated by practical examples with RedPin and BluePin.

Figure 4 shows the configuration template of InL-Adapter.

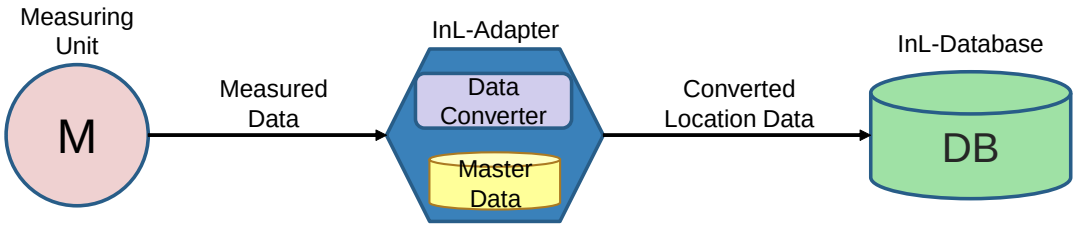


Figure 4: Configuration of InL-Adapter

As seen in the figure, the role of the InL-Adapter is to convert the measured location data in a proprietary format into the one in DM4InL. To achieve the conversion, two kinds of data depending on IPS are essential: *measured data* and *master data*. The measured data is real-time location data (i.e., raw data) measured by the given IPS. The master data is static data specifying various configuration information of IPS, such as users, devices, stations, buildings and indoor maps. As shown in Figure 4, every InL-Adapter contains *data converter*, which defines a specific mapping from the measured data into DM4InL based on the master data.

To help understanding, we demonstrate the process of data conversion of BluePin and RedPin, using instances of location data L1 and L2 (See Section 2.1). According to DM4InL, we divide data items in L1 (or L2) into three elements: time, object, position. As for the time information, it is easy to introduce the common representation in UTC.

For example, `2015/07/27 11:23:45 JST` in L1 can be converted into `2015-07-27T02:23:45`. For L2, since RedPin does not define time attribute, we need to modify the RedPin client to add the times tamp to L2, like `2015-07-27T01:11:01`.

As for the object information, we create a mapping from a proprietary ID of the located object into an object ID. For instance, the person ID `niu` in L1 of BluePin is bound to object ID of DM4InL. Using the master data of BluePin, other data items of the object model can be filled. On the other hand, the macAddress in L2 of RedPin can be mapped to the object ID of DM4InL, since it is a unique string.

The conversion of the position information is rather complex. For L1, we need to convert the symbolic information "22" and "S101 Entrance" into a spot in building model of DM4InL. Also, the spot should be represented by a local point. For this, we use the master data of BluePin to look up the detailed location information of `22`. Suppose that the detailed information points position (3.50m, 5.5m, 1.5m) of a building `B001`. Then we create a spot "S101 Entrance" in `B001`, whose coordinate is (3.50, 5.50, 1.50). Finally, we define a mapping from L1 to the spot.

On the other hand, as seen in L2, RedPin represents the position based on 2-dimensional coordinate over a given map, i.e., image of the floor plan. Therefore, multiplying the coordinates by the map scale derives the actual X and Y offsets. The Z offset can be derived from the altitude of the floor. Thus the coordinates of a local point can be calculated. The spot information can be derived from meta-data of the floor ma. For instance, suppose that `KU-System-1F` represents a map of the first floor of building B001 with altitude of 1.5m, and that the map scale is 1/51.6. Then, L2 is converted into a spot bound to a local point (6.70, 10.44, 1.50).

Based on the above conversion, the heterogeneous measured data L1 and L2 are converted into DM4InL format showed in Table 1, 2, 3.

Table 1: LocalPoint

Pcode	x-offset	y-offset	z-offset	Building-Seq
P001	3.50	5.50	1.50	B001-01
P002	6.70	10.44	1.50	B001-02

Table 2: Spot

BuildingID	SpotID	SpotName	PointCode
B001	s00001	S101-Entrance	P031
B001	s00002	S103	P001

Table 3: ObjectLocationLog

ObjectID	P-Code	DateTime
niu	P001	2015-07-27T02:23:45
08:60:6e:32:b6:0b	P002	2015-07-27T01:11:01

We have developed an InL-Adapter for RedPin and modified the client of RedPin. The modification of RedPin Android client comprised of around 418 lines of code, and the InL-Adapter of RedPin comprised of around 536 lines of code. Technologies used for the implementation are as follows: **Language:** Java 1.7.0, **Database:** MySQL 5.1, **Web server:** Apache Tomcat 7.0.57, **Web service engine:** Apache Axis 2 1.6.2.

5. INL-QUERY FOR OPERATION INTEGRATION

5.1 Overview

To achieve the operation integration, the proposed CIF4InL implements InL-Query (Indoor Location Query Service). InL-Query is a Web service that provides application-neutral API for querying indoor location data stored in InL-Database. It is supposed to be deployed on cloud.

According to the data schema of DM4InL, we develop two types of API for InL-Query: *fundamental API* and *composite API*, as mentioned in Section 3.2 The fundamental API provides interface for querying entities within a signal model at a time: location, building or object model. The details will be described in Section 5.2. The composite API allows advanced queries accessing multiple models simultaneously, which will be explained in Section 5.3.

5.2 Fundamental API

DM4InL represents location information of every indoor object with three kinds of models: location, building and object. (See Section 3.3). Depending on model to which a given query belongs, we define three groups of API: *location query API, building query API* and *object query API.*

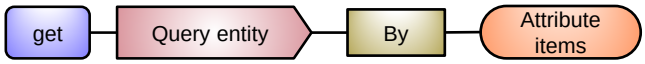


Figure 5: Derivation process of building and object query API

Table 4: Detail Entity and Attribute Item Table

QUERY ENTITY	ATTRIBUTE		ITEM
BUILDING	Theme Attribute		BuildingID, Name, Type
	Spatial Attribute		GPID
PARTITION	Theme Attribute		PartitionID, Name
	Spatial Attribute		SpaceCode, BuildingID
ROUTE	Theme Attribute		RouteID, Name
	Spatial Attribute		LineCode, BuildingID
SPOT	Theme Attribute		SpotID, Name
	Spatial Attribute		PointCode, BuildingID
OBJECT	Theme Attribute	NULL	ObjectID, Type
		People	ObjectID, name, sex, ...
		Appliance	ObjectID, Appliance Type, status, ...
	Spatial Attribute		PointCode
	Time Attribute		Date Time

5.2.1 *Location Query API*

This API provides a set of methods (i.e., functions) querying any entity within the location model. As shown in Figure 2, the location model consists of four entities. Each entity has a set of methods that returns appropriate instances based on given known attributes. The naming convention of the methods is `get[TargetEntity]By[given attribute]`. For instance, `getLPByPointCode(pointCode)` returns a local point designated by the given point codes.

The local query API also provides methods querying *spatial relation* among geometric primitives in location model. The spatial relation can be used to investigate how a spatial object in a space is located in relation to another object. In Location Query API, we define two types of spatial relation:

- **Topological relation:** It represents how an object is topologically related to another object. The operators manipulating the topological relation include: `within`, `covers`, `coveredBy`, `intersects`, `touches`, `equals`, `disjoint`, `crosses`, `overlaps`. For instance, `getLPwitninLSP(LocalSpace)` returns all the local points within a given certain local space.
- **Distance relation:** It represents how far an object is from another object. The operators manipulating the distance relation include: `at`, `nearby`, `vicinity`, `far`. For instance, `getLPnearbyLP(LocalPoint)` returns all local points nearby a given local point.

5.2.2 *Building Query API*

This API provides methods for querying entities defined in building model. In order to cover all possible queries, we derive the methods based on the structure shown in Figure 5. The figure shows that every method is constructed by varying *query entity* and *attribute items*. The query entity represent an entity to be returned by the method. Each entity has a designated set of attribute items, which explain the entity from the theme or spatial perspectives. Table 4 summarizes the entities and attribute items contained in the building model (as well as in the object model). The methods are derived from all the possible combinations of the entities and the attribute items.

For instance, `getBuildingByGPID(GPID)` returns a building identified by a given global position ID. Also, `getRouteByName(buildingID, spotname)` searches routes in a given building by its name.

5.2.3 *Object Query API*

This API provides methods querying entities in the object model. Similar to the Building Query API, the methods are constructed based on the structure in Figure 5. The attribute items are shown in the bottom of Table 4. We can see that the time attributes exists for the object entity, as an object usually take different location as the time passes. The methods are derived from all the possible combinations of the entities and the attribute items.

For instance, `getObjectsAtPoint(pointcode, dateTime)` returns a set of object that exist in a given local point on given data and time.

5.3 Composite API

The fundamental API allows only basic queries limited within a signal model. Hence, it is often too primitive to meet sophisticated requirements of InL-App, which requires developers to integrate multiple API manually. For example, to implement a query "Who is in Room S101?", the developer need to integrate the building query API and the object query API. This motivated us to develop the *composite API*, which allows high-level queries by internally combing some fundamental API.

Methods of the composite API have been derived based on typical use cases of location query in InL-App, so that they can reduce development cost and effort. Investigating the typical use cases, we have developed three type of composite API. The first type is *building−object* API, which returns geographic elements of a building based on known information of an object. For instance, `getPartitionContainObject(objectID)` returns a partition that contains a given object. The method is implemented by re-using multiple methods of the fundamental API, specifically:

```
1 Partition getPartitionContainObject(objectID)
2  Object o=geObjectByID(objectID)
3  LocalSpace ls=getLSPContainsLP(o.pointCode)
4  Partition p=getPartitionByCode(ls.spacecode)
5    return p
```

The second type is *object−building* API, which searches objects based on known geographic elements of building. For instance, `getPeopleWithinPartition(bName, pName)` returns

people within a partition `pName` of a building `bName`. This method can be implemented as follows:

```
Person [] getPeopleWithinPartition(bName,
 pName)
 Partition p=getPartitionByName(bName, pName)
 LocalSpace ls=getLSPBySpaceCode(p.spaceCode)
 LocalPoint[] lps = getLPcontainedInLSP(ls)
 Person [] H = empty
 foreach ls in lps
  Person h=getPersonAtPointIntim(
  lp.pointCode, NOW)
  push(H, h)
 return H
```

The last type is *calculation API*, which measures a certain metric among object and geographic elements, using the spatial relations. For instance, `getDistanceBetweenSpotAndObject(buildingID, spotId, objectId)` returns a distance between a given spot and a given object. This method can be implemented as follows:

```
double getDistanceBetweenSpotAndObject(
 buildingId, spotId, objectId)
 Spot s=getSpotBySpotId(buildingId, spotId)
 Object o=getObjectByObjectId(objectId)
 double d = getDistanceBetweenLP(
   s.pointcode, o.pointcode)
 return d
```

The implementation of the API is currently under way. Technologies used for the implementation are as follows:

- **Language:** Java 1.7.0,
- **Database:** MySQL 5.1,
- **Web server:** Apache Tomcat 7.0.57,
- **Web service engine:** Apache Axis 2 1.6.2.

6. EVALUATION

To evaluate the practical feasibility of CIF4InL, we conduct a comparative study among three IPS: RedPin, BluePin and CIF4InL (that integrates RedPin and BluePin).

6.1 Capabilities for Location-Dependent Queries

The comparison is based on sufficiency of *essential capabilities of location-dependent queries* [5]. The location-dependent means that any change of the locations of an object significantly affects the result of query for the object. For example, suppose that a user A wants to find friends within a range of 100m from A while navigating a shopping center. The result of the query depends on A's current position, as well as on the location of the friends. According to [5], the following capabilities should be supported especially in the indoor location queries:

- **Position Queries** return the locations of mobile and static objects, and are processed according to either a geometric or a symbolic model of space.
- **Navigation Queries** encompass all queries that directly help the users to find and reach some points of interest by providing them with navigational information, while optimizing some criteria such as total traversed distance or travel time.
- **Range Queries** are used to find and retrieve information on objects of interest or places within a user-specified range or area.
- **k Nearest Neighbor(kNN) Queries** search for the k closest qualifying objects to a moving user with respect to his or her current location.

Moreover, Liu et al. [10] suggested that the time is also an essential attribute for the location-dependent query.

- **Time queries** search a target object and a location by time, or retrieves the time from an object and a location. Each query depends on a record that the object stayed in the location at the time.

6.2 Result of Comparison

Table 5 compares the three IPS with respect to the above five capabilities. In the table, labels ◯, △ and × represent that the capability is "satisfied", "partially satisfied", and "not satisfied", respectively. First, we can see that all the three IPS supports the position queries. Although their representations of the position are different, they all have methods to ask the indoor location of a target object.

The navigation queries cannot be supported by RedPin or BluePin. In RedPin and BluePin, no topology information among multiple locations is not maintained. Also, it cannot be derived from individual location data, since each location is represented as a pre-defined venue (not a position). However, once CIF4InL converts their location data into DM4InL, the topology can be defined using the coordinates of the locations. Using the topology, CIF4InL can support applications to implement navigation queries. Note, however, that the queries are limited to the spots or positions that are already registered in InL-Database.

CIF4InL binds indoor location with a three-dimensional coordinate in DM4InL. Using the topological and distance relations and the calculation API, one can easily implement the range queries. However, the range queries cannot be supported by BluePin, since BluePin specifies each location as a symbolic label, from which we cannot calculate the range. The location data of RedPin contains a two-dimensional coordinate on the map, from which we can calculate the distance between two locations. However, when two locations are represented in separate maps (e.g., different floors), the distance cannot be calculated. In that sense, RedPin cannot fully satisfy the range queries. The same discussion applies to the kNN queries, since the essentials of the kNN queries are almost the same as the ones of the range queries.

As for time query, RedPin cannot support them as the data does not contain any time attribute, BluePin contains a time-stamp in the location data, while CIF4InL manages time-series data of ObjectLocationLog in DM4InL. Therefore, these two IPS can support the time queries.

Based on the above discussion, we can see that CIF4InL supports more capabilities for the location-dependent queries. Through the data and operation integration, CIF4InL even enriches the existing proprietary IPS. Thus, it is expected that application developers can develop InL-App more efficiently and intuitively, using CIF4InL.

7. RELATED WORK

Our work is situated closely to the intersecting fields of indoor location framework, indoor positioning platforms, and

Table 5: Comparison of three IPS w.r.t. capabilities of location-dependent queries

IPS	Position	Navigation	Range	kNN	Time
RedPin	◯	×	△	△	×
BluePin	◯	×	×	×	◯
CIF4InL	◯	△	◯	◯	◯

data modeling techniques for indoor spaces. A number of indoor positioning framework or platforms have been proposed so far.

Brachmann et al. [2] proposed a multi-platform software framework. It aims to manage sensor data from different smartphone platforms for better understanding of RSSI(Wi-Fi)-based and other sensor-based IPS. The key idea of this framework lies in the normalization techniques for individual sensor data, such as magnetometer, accelerometer and gyroscope. Thus, the framework is limited for wireless IPS, which belongs to indirect self-positioning system (see Figure 3-(4)). It does not consider other IPS topology. In this sense, the application scope is narrower than CIF4InL.

Gubi et al. [3] presented a platform that can dynamically provide efficient location technologies. As a user moves around a building, the platform suggests a best-available indoor positioning method based on current position of the user. The platform manages building data in the form of symbolic map, and markup of associated RF infrastructure Wi-Fi and Bluetooth. However, the platform assumes that applications manage their own maps individually. So, it does not provide application-neutral API that can re-use the indoor location data over different applications.

INSTEO Inc. [6] present an IPS technology that relies on optimal hybridization algorithms of multiple information sources. The data source includes power measurement of Wi-Fi, Bluetooth Low Energy signals, smartphone sensors (accelerometer, compass, barometer, and so on). However, this approach is similar to Gubi's, which focuses on the combination of location technologies at the IPS level. Neither of them aims the loose coupling between IPS and InL-App.

8. CONCLUSION

In this paper, we have proposed a cloud-based integration framework, called CIF4InL, in order to achieve data and operation integration for heterogeneous IPS. To achieve the data integration, CIF4InL implements InL-Adapter which provides different adaptation patterns for different system topology of IPS. We also have implemented InL-Query, which provides fundamental API and composite API based on the data schema of DM4InL. CIF4InL contributes to loose coupling of IPS and InL-App, which will significantly improve the efficiency and re-usability in the InL-App development. We have actually applied the proposed framework to integrate two existing IPS, RedPin and BluePin. In addition, we have evaluated the CIF4InL by investigating the sufficiency of the five capabilities of location-dependent queries.

As for the future work, we plan to conduct further evaluation of CIF4InL, with respect to performance and security for practical use cases. We are also interested in how to address more pragmatic issues. They include *uncertainty* of location caused by unreliable devices, as well as *feature interactions* when integrating data and operations.

9. 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)].

10. REFERENCES

[1] P. Bolliger. Redpin - adaptive, zero-configuration indoor localization through user collaboration. In *Proceedings of the First ACM International Workshop on Mobile Entity Localization and Tracking in GPS-less Environments*, MELT '08, pages 55–60, September 2008.

[2] F. Brachmann. A multi-platform software framework for the analysis of multiple sensor techniques in hybrid positioning systems. In *Proceedings of 10th Conference on Telematics Engineering*, JITEL 2011, September 2011.

[3] K. Gubi, R. Wasinger, M. Fry, J. Kay, T. Kuflik, and R. Kummerfeld. Towards a generic platform for indoor localisation using existing infrastructure and symbolic maps. In *Proceedings of 18th International Conference on User Modelling, Adaptation and Personalisation*, June 2010.

[4] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster. The anatomy of a context-aware application. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, MobiCom '99, pages 59–68, 1999.

[5] S. Ilarri, E. Mena, and A. Illarramendi. Location-dependent query processing: Where we are and where we are heading. *ACM Comput. Surv.*, 42(3):12:1–12:73, March 2010.

[6] INSITEO Inc. Platform of insiteo, 2014. Retrieved September 7, 2015 from http://www.insiteo.com/joomla/index.php/en/plateform.

[7] Y. Kashio, S. Matsumoto, S. Saiki, and M. Nakamura. Design and implementation of service framework for presence sensing in home network system. In *The Third International Conference on Digital Information, Networking, and Wireless Communications*, DINWC2015, pages 109–114, February 2015.

[8] N. Kothari, B. Kannan, E. D. Glasgwow, and M. B. Dias. Robust indoor localization on a commercial smart phone. *Procedia Computer Science*, 10:1114–1120, August 2012.

[9] J. A. B. Link, P. Smith, N. Viol, and K. Wehrle. Footpath: Accurate map-based indoor navigation using smartphones. In *Proceedings of 2011 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–8, September 2011.

[10] H. Liu, H. Darabi, P. Banerjee, and J. Liu. Survey of wireless indoor positioning techniques and systems. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 37(6):1067–1080, November 2007.

[11] K. Lorincz and M. Welsh. Motetrack: A robust, decentralized approach to rf-based location tracking. In T. Imielinski and H. F. Korth, editors, *Location- and Context-Awareness*, pages 63–82. Springer Berlin Heidelberg, 2005.

[12] D. Manandhar, S. Kawaguchi, and H. Torimoto. Results of imes (indoor messaging system) implementation for seamless indoor navigation and social infrastructure platform. In *Proceedings of the 23rd International Technical Meeting of The Satellite Division of the Institute of Navigation*, ION GNSS 2010, pages 1184–1191, September 2010.

[13] M. K. Mohamed, S. Patra, and A. Lanzon. Designing simple indoor navigation system for uavs. In *Proceedings of 19th Mediterranean Conference on Control Automation (MED)*, pages 1223–1228, June 2011.

[14] L. M. Ni, Y. Liu, Y. C. Lau, and A. P. Patil. Landmarc: indoor location sensing using active rfid. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom 2003)*, pages 407–415, March 2003.

[15] L. Niu, S. Matsumoto, S. Saiki, and M. Nakamura. Considering common data model for indoor location-aware services. In *Proceedings of the 4th International Workshop on Location and the Web*, LocWeb '14, pages 25–32, November 2014.

[16] P. Tarrio, M. Cesana, M. Tagliasacchi, A. Redondi, L. Borsani, and J. R. Casar. An energy-efficient strategy for combined rss-pdr indoor localization. In *Proceedings of 2011 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pages 619–624, March 2011.

[17] S. P. Tarzia, P. A. Dinda, R. P. Dick, and G. Memik. Indoor localization without infrastructure using the acoustic background spectrum. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, pages 155–168, June 2011.

[18] R. Wand, A. Hopper, V. Falcao, and J. Gibbons. The active badge location system. *ACM Trans. Inf. Syst.*, 10(1):91–102, January 1992.

[19] K. Watanabe. *Introduction to Data Modeling for Database Designing*. Nippon Jitsugyo, Tokyo, 2003.

[20] M. Yuan. Wildfire conceptual modeling for building gis space-time models. In *Proceedings of GIS/LIS '94, Annual Conference and Expo Phoenix, Arizona*, GIS/LIS '94, pages 860–869, October 1994.

[21] J.-C. Zufferey, A. Klaptocz, A. Beyeler, J.-D. Nicoud, and D. Floreano. A 10-gram microflyer for vision-based indoor navigation. In *Proceedings of 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3267–3272, October 2006.