MashMap: Application Framework for Map-based Visualization of Lifelog with Location

Kohei TAKAHASHI, Akira SHIMOJO, Shinsuke MATSUMOTO, Masahide NAKAMURA
Graduate School of System Informatics, Kobe University, JAPAN
1-1 Rokkodai-cho, Nada-ku, Kobe, Hyogo 657-8501, Japan
Email: {koupe, shimojo}@ws.cs.kobe-u.ac.jp, {shinsuke, masa-n}@cs.kobe-u.ac.jp

***Abstract*—This paper presents an application framework, called *MashMap framework*, which facilitates development of location-based lifelog services. The proposed framework imports various types of location log data, and stores them in a database with the Life Log Common Data Model (LLCDM). A developer first creates a data source consisting of a filter and a display format. The filter extracts necessary data from the database, while the display format defines how the data is shown on the map. The developer then defines a MashMap by choosing a single or multiple data sources. The MashMap is an object, which aggregates several data sources on a single map, shown in the designated display format. The object is finally visualized on a Google Map by MashMap renderer. We conduct a case study developing a travel log review service and a conference history map using MashMap framework.**

I. Introduction

Lifelog is a social act to record and share human life events in open and public form [1]. A variety of *lifelog services* currently appear in the Internet. Using these services, we can easily store, publish and share various types of lifelogs on the Web. Due to spread of smartphones and GPS loggers, many lifelog services extensively use *location information* associated to lifelog records (i.e., *lifelog with location*). Popular services include *GARMIN* for managing training data, *foursquare*[2] for sharing the "check-in" places. Moreover, the conventional services like *Twitter* and *Facebook* have now implemented features attaching location information to their lifelogs.

The lifelog with location enables users to look back what happened when and *where*. To review the "where aspect" efficiently, most services introduce *map-based visualization*, in which the lifelog data with location are displayed on a map. Basically, the map-based visualization has been developed by different vendors for different services. In the visualization, however, quite a few of processes (e.g., extracting location, plotting a point on a map, etc.) do not rely on specific kinds of services, Indeed, the location information is a common data independent of kinds of lifelog[3] This motivated us to consider a *common framework* to implement the map-based visualization of lifelog. The framework would help rapid and reliable development of the location-based lifelog services.

In this paper, we present an application framework, called *MashMap framework*, witch visualizes various kinds of lifelogs on the Google Maps. In the *MashMap framework*, various kinds of lifelogs with location are first transformed into the lifelog common data model (LLCDM)[3] and stored in a database.

Then, an application developer defines a *filter* and *display format*. The filter extracts lifelog dataset that is visualized from the database. The display format defines a format of how the extracted dataset should be displayed on a map. A pair of the filter and the display format specifies a *data source*. Next, the developer creates a *MashMap* object as an aggregation of one or several data sources. The *MashMap* object is a data object that specifies a group of data sources, each of which should be visualized in a designated format on a map. Finally, the *MashMap* object is displayed on a map by a *MashMap renderer*, implemented as JavaScript library. The proposed framework does not require any expertise of the GoogleMaps API or proprietary lifelog data format. Thus, the developers can create location-based lifelog services more rapidly and easily.

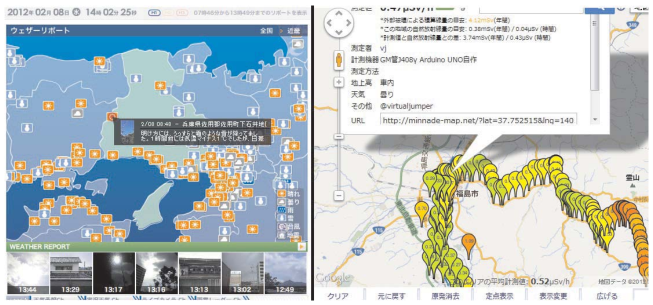
We have implemented the *MashMap* framework by JAX-RS (Jersey: JAX-RS (JSR 311) Reference Implementation for building RESTful Web services.[4]) deployed on Google App Engine. To demonstrate the effectiveness, we have implemented MashMap Viewer using the developed MashMap. We have also created MashMap objects of Travel Log map, and Event Venues Log map. It was shown that the proposed framework significantly reduced the development effort to visualize location-based lifelog services.

II. Preliminaries

A. Lifelog with Location

In this paper, we call lifelog records containing location information *lifelog with location*. Each lifelog with location is associated with date, time and location of an event, related users, etc. Specifically, the location information consists of a pair of *[latitude, longitude]* or an *address*. Since the location information is generic data, it can be associated with various kinds of lifelogs (e.g., pictures, tweets, motion logs, etc.) [3].

Due to spread of smartphones equipped with GPS and inexpensive GPS loggers, the location information can be



a. Weather Report Ch. b. Radiation Dose Data Gathering Map

Figure 1. Examples of Web Services using Lifelog with Location

acquired easily. The lifelog with location has information of who, when, where, what happened, which characterizes human acts specifically. Thus, the lifelog with location is expected to be used various application area, including QoL improvement, entertainment, traffic, logistics, agriculture, disaster prevention, and so on.

B. Application Services Using Lifelog with Location

There are many web services using the lifelog with location. For example, Figure 1(a) is a screenshot of Weather Report Ch.[5]. This service provides not only official weather information of major cities, but pinpoint news reported by end users, called reporters. Using a smartphone, a reporter reports weather information with pictures, comments, and location. The log is collected around the country, aggregated on a server, and visualized within the same map. By clicking a marker on the map, a user can check a pinpoint weather report with pictures or comments.

Next, Figure 1(b) is a screenshot of Radiation Dose Data Gathering Map[6]. This service was developed after the Fukushima Daiichi nuclear disaster. Individual users measure an amount of radiation, and send it with location to the server. Logs of radiation levels are visualized on the same Google Map. Gathering logs, global map of radiation levels is created. By clicking a marker on the map, user can check a radiation level, measured date, used tool at the point.

There are many other services using lifelog with location, including *GARMIN*, *foursquare*[2], *Twitter*, *Facebook*, etc.

C. Lifelog Mashup Platform (LLCDM, LLAPI)

To support efficient lifelog mashup, we have previously proposed the *lifelog common data model(LLCDM)* and *lifelog mashup API(LLAPI)*[3] [7]. Table I shows the data schema of the LLCDM. We derived data items essential for lifelog, from the viewpoints of what, why, who, when, where and how. LLCDM standardizes information most lifelogs should have (e.g., date, time, user, location, etc.). For application-specific data (in WHAT perspective), LLCDM delegates the interpretation of the data to external schema.

LLAPI specifies interface for searching and retrieving lifelog data conforming to the LLCDM. The following

Table I
COMMON DATA SCHEMA OF LLCDM

perspective	**data items**	**description**
WHEN	`<date>`	Date when the log is created
	`<time>`	Time when the log is created
WHO	`<user>`	Subjective user of the log
	`<party>`	Party involved in the log
	`<object>`	Objective user of the log
WHERE	`<location>`	Location where the log is created
HOW	`<application>`	Service/application by which the log is created
	`<device>`	Device with which the log is created
WHAT	`<content>`	Contents of the log (whole original data)
	`<ref_schema>`	URL references to external schema
WHY	`n/a`	n/a

shows an API that returns lifelog data matching a given query related LLCDM's data column.

```
getLifeLog(since, until, user, location,
  application, select)
```

LLAPI is published as Web service (REST, SOAP) and can be invoked from various platforms.

D. Challenge on Map-based Visualization

As seen in Section II-B, most lifelog services with location information visualize their data onto a map. Although the visualization process is quite similar to each other, it is developed by individual service provider, and is not reused at all. In addition, features using location information vary from one service to another.

Our interest here is to establish a common *application framework* that can visualize various kinds of lifelog with location. The common framework enables rapid and reliable development of map-based lifelog services. Moreover, we want to *mashup* different of data onto the same map, in order to achieve more value-added services.

III. APPLICATION FRAMEWORK FOR MAP-BASED VISUALIZATION OF LIFELOG WITH LOCATION

A. MashMap framework

To cope with the challenges, we develop a novel application framework, called *MashMap*, in this paper. "MashMap" is a coined word meaning to *mash up* various lifelogs on a *map*. In this framework, a developer defines data source (*DataSources*) by selecting necessary data from various kinds of lifelogs, and then creates a map (*MashMap*) in which several data sources are integrated.

Figure 2 shows the overall structure of *MashMap* framework. The top of the diagram shows various lifelogs with location recorded by users. These lifelogs are imported to a database of the proposed framework according to needs of

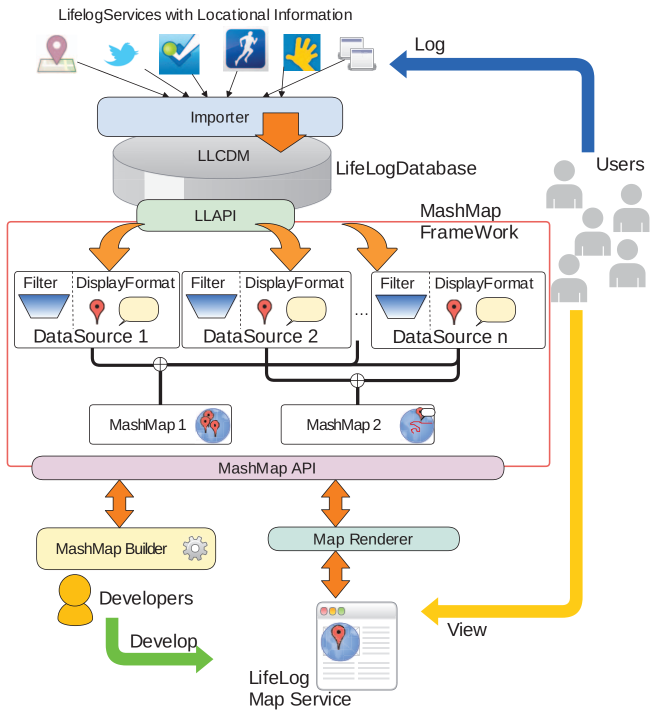


Figure 2. Overall structure of MashMap framework

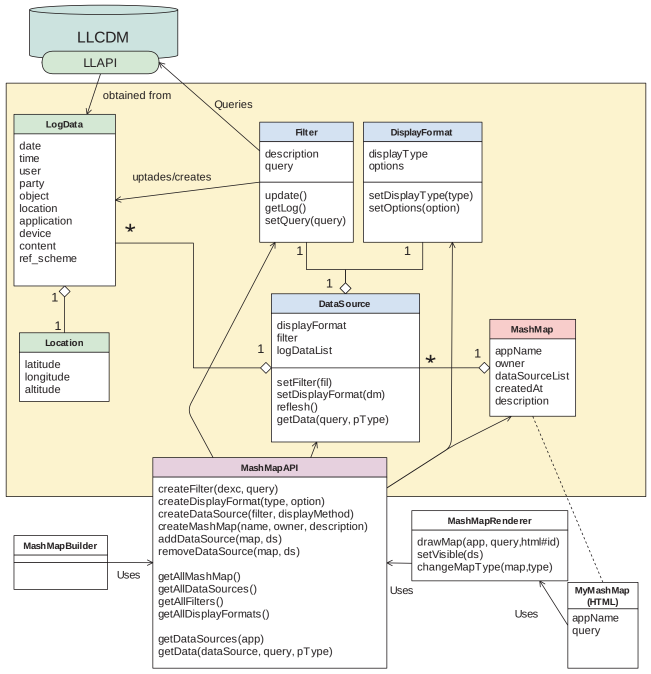


Figure 3. Class diagram of MashMap framework

users. Once logs are imported, they are transformed to the *LLCDM* format accessed via LLAPI in Section II-C.

An application developer creates *DataSources* by selecting necessary lifelogs stored in the database. For this, it is necessary to define `Filter` and `DisplayFormat`. `Filter` specifies a condition for selecting data. `DisplayFormat` specifies a way how to visualize the data on a map.

Next, the developer creates a *MashMap* object, and connects the created `DataSources` to it. Finally, the `MashMap` object is visualized as a map by `MashMap Renderer`. On the map, each *DataSources* of the *MashMap* is shown in defined *DisplayFormat*. Thus, the map on which various lifelogs are integrated is created.

The creation of `MashMap` object and configuration of `DataSource` are all performed via MashMap API. By deploying the core framework (shown in middle of Figure 2) in the cloud, we can make `MashMap` framework as SaaS (Software as a Service).

Figure 3 shows a class diagram of the proposed framework. Each class is explained in the following subsections.

B. LogData

As mentioned in the previous section, all lifelogs with location used in the MashMap framework is standardized in the LLCDM format, and stored in the database. In the framework, the data is searched and retrieved by `getLifeLog()` LLAPI introduced in Section II-C. Each data record imported to the MashMap framework is stored in an object of the `LogData` class. Attributes of the `LogData` class is based on the *LLCDM* format. For example, a lifelog record of "user koupe accessed to the foursquare from iPhone4 and checked in Bizan Park at 16:25, on September 11th, 2011." is expressed as the following `LogData` object.

```
[2011-09-11, 16:25:00, "koupe", --, --,
 [34.066987, 134.532727, 256.7244873046],
 foursquare, iPhone4, "...shout:climb and
 climb... @ Bizan Park"...,https://ja.
 foursquare.com/v/%E7%9...]
```

C. Filter

The `Filter` class is for selecting necessary data from a database. A developer using the framework creates a `Filter` object and sets the query to retrieve necessary data. Based on the query, the `Filter` object executes LLAPI, obtains the data from the data base, and import the records as `LogData` objects. The query set to the `Filter` object is based on the parameters of `getLifeLog()`.For example, the following filters select koupe's lifelogs recorded by *foursquare* and *GARMIN* on September 11th, 2011 into separate datasets. In the following, "*" denotes a wildcard.

```
filter1= [description:"2011/09 foursqu-
   are in Tokushima pref.", query:{2011-
   09-11, 2011-09-12, 00:00:00, 00:00:00,
   koupe, *, "foursquare", *, *)}]
filter2= [description:"2011/09 GARMIN-
   logs in Tokushima pref.", query:
   {2011-09-11, 2011-09-12, 00:00:00,
    00:00:00, koupe, *, "GARMIN", *, *)}"]
```

D. DisplayFormat

The `DisplayFormat` class defines how the lifelog data is displayed on the map. Each kind of lifelog data has a suitable format to be visualized on a map. For example, in the case of a GPS trail recorded by GARMIN, a line connecting the points looks good. In the case of a tweet of Twitter, a marker with a balloon would be suitable.

Even for the same data, the suitable display format may vary depending on applications and other data integrated with. So, it is useful for the dataset to freely change the own format on a map, which achieves effective visualization. Each `DisplayFormat` object has two attributes: format type and an optional setting of a map. A format type is either of a marker (Marker), an icon (Icon), a line (Polyline) or a balloon (InfoWindow). An optional setting specifies options for GoogleMaps, given in JSON format based on GoogleMaps JavaScriptAPI V3.

For example, a `DisplayFormat` object whose type is a marker without shadow is given by

```
 disp_marker = [Marker, "flat:true"]
```

Another example, where the object type is a red line, weight is 2px, and opacity is 100%, is specified by

```
 disp_line = [Polyline, "strokeColor:#F00,
  strokeWeight:2, strokeOpacity:1"]
```

E. DataSource

The `DataSource` specifies an interesting data set for the developer. Each `DataSource` object has references to a `Filter` and a `DisplayFormat` objects. When `getData()` method is called, `LogData` objects are retrieved from the database by using the corresponding `Filter` object. It is also possible to filter the retrieved data again by giving a query to `getData()`. This is usually used to limit the number of data on the map at runtime.

Retrieved lifelog data is supposed to be shown based on the specified `DisplayFormat`. Thus, each `DataSource` object defines an atomic sequence of data to be visualized on a map. Also the user can turn on or off each data source on the map. Using the previous examples, if we show lifelogs taken by foursquare as a marker, the data source will be:

```
ds_foursq = [filter:filter1,
          displayType:disp_marker,
          logDataList:[...]]
```

Also, the `DataSource` showing logs of *GARMIN* as a line is following.

```
ds_garmin = [filter:filter2,
          displayType:disp_line,
          logDataList:[...]]
```

F. MashMap

The `MashMap` class defines a MashMap object, which aggregates several DataSources and overlaps them over a map. It also contains meta-information like map name, author, created date, description, etc. An example of MashMap object would be "Travel Log Map" aggregating GPS trails, tweets, pictures. Another example is "Weather Map" consisting of pin-point weather reports and user's real-time comments. Using the same example in the previous sections, we can define "Log of one-day trip to Tokushima pref." as the following MashMap object.

```
["2011/09 Log of one-day trip to
 Tokushima pref.", "koupe", [ds_foursq,
 ds_garmin], 2012-01-03 07:05:12,
 "one-day trip to Tokushima pref..
 created by GARMIN and foursquare logs"]
```

G. MashMap API

The `MashMap API` is a facade class to provide CRUD operations (create, read, update, and delete) of various objects within the MashMap framework. With the API, external users and software agents defines own filters, display formats, datasources, and `MashMap` objects. It is also possible to obtain datasets of `DataSource` objects.

The following pseudo code represents a recipe for creating a new MashMap of "Log of one-day trip to Tokushima pref.", using `MashMap API`.

```
/* Create a MashMap */
tokushima = createMashMap("Log of
  one-day trip to Tokushima pref.",
  "koupe", "2011/09 one-day trip
  to Tokushima pref.");
/* Create a DataSource of GARMIN */
f1 = createFilter({user:koupe,
      application:GARMIN,
      s_date:2011-09-11,...});
d1 = createDisplayFormat(Polyline,
      {strokeColor: #FF0000, ...});
garmin = createDataSource(f1, d1);
/* Create a DataSource of foursquare */
f2 = createFilter({user:koupe,
      application:foursquare,
      s_date:2011-09-11,...});
d2 = createDisplayFormat(Marker,
      {flat:true,...});
fsq = createDataSource(f2, d2);
/* add two DataSources to a MashMap */
addDataSource(tokushima, garmin);
addDataSource(tokushima, fsq);
```

H. MashMap Renderer

The `MashMap Renderer` is a JavaScript library that visualizes a MashMap object onto the Google Map. A developer passes an id of a MashMap object as a parameter. Then, `MashMap Renderer` obtains the MashMap object and associated data sources. Finally, it draws the data sources (i.e., lifelogs with location) in the designated display format, by invoking appropriate Google Maps API [8].

I. MashMap Builder

The `MashMap Builder` is a GUI frontend of MashMap API for developers. It allows a developer to create a MashMap object with easy and intuitive operations on the screen. Using the builder, a developer defines own `Filter`, `DisplayFormat` and `DataSource` objects, to construct a `MashMap` object. It is also possible to reuse several ready-made definitions.

MashMap Builder also has a feature to export a minimal skeleton of an HTML file, implementing a Web application with the created MashMap. The developer customizes the skeleton, according to the purpose and features of the target service.

J. Implementation

We have implemented the MashMap framework. Table II shows a technology stack used for the implementation. The MashMap API is implemented as *RESTful Web services*, and the core framework has been deployed on *Google App Engine*. This allows the MashMap framework to work as SaaS cloud service. The total lines of code is 2160, and the development effort was 1.5 man-month.

Table II
TECHNOLOGIES USED TO DEVELOP MASHMAP FRAMEWORK

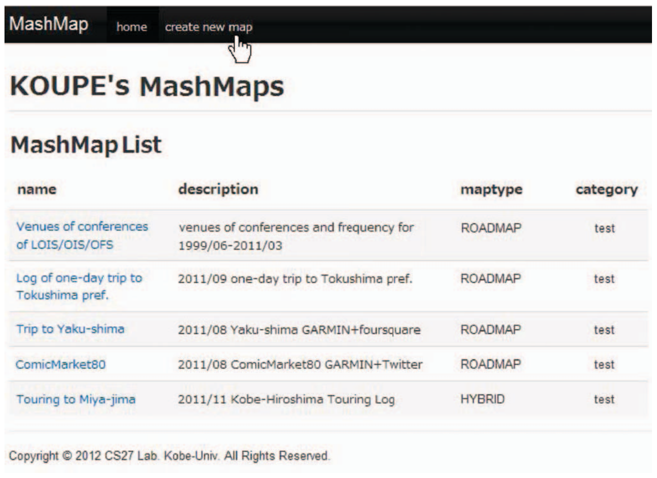
Platform	Google App Engine for Java
Language	Java, HTML, JavaScript
API middleware	JAX-RS RESTful Web Service
JavaScript Library	Google Maps JavaScript API V3 jQuery1.7.1
CSS Framework	Twitter BootStrap v1.4.0
jQuery Plugin	Twitter BootStrap Dropdown Twitter BootStrap Tabs DatePicker
Operation Checked Browser	Google Chrome 19.0

IV. CASE STUDY

Using the proposed framework, we develop several practical Web applications.

A. MashMap Viewer

The application `MashMap Viewer` provides a simple viewer of created MashMaps. Figure 4 shows the screenshot. In this figure, all MashMaps created by a user *koupe* are listed. When a user clicks an entry, the map view page is shown. In the page, a user can toggle the display of each DataSource on the map. Also, the user can narrow the range of displayed lifelogs by year, month, or day.



KOUPE's MashMaps

MashMap List

name	description	maptype	category
Venues of conferences of LOIS/OIS/OFS	venues of conferences and frequency for 1999/06-2011/03	ROADMAP	test
Log of one-day trip to Tokushima pref.	2011/09 one-day trip to Tokushima pref.	ROADMAP	test
Trip to Yaku-shima	2011/08 Yaku-shima GARMIN+foursquare	ROADMAP	test
ComicMarket80	2011/08 ComicMarket80 GARMIN+Twitter	ROADMAP	test
Touring to Miya-jima	2011/11 Kobe-Hiroshima Touring Log	HYBRID	test

Copyright © 2012 CS27 Lab. Kobe-Univ. All Rights Reserved.

Figure 4. MashMap Viewer

MashMap Viewer is written in HTML, JavaScript, and JavaServlet. The total lines of code is just 680. Most features required for implementing the viewer are provided by the proposed framework as they are. Thus, we can see that the proposed framework contributes to significant reduction of development effort.

B. Travel Log Map

We implemented the example of *Log of one-day trip to Tokushima pref.* in Section III, using the MashMap framework. Figure 5 is screenshot of the map displayed by the MashMap Viewer. The figure shows an actual travel logs recorded by the author during one-day trip to Tokushima City, Tokushima pref. on September 11th, 2011. A red line shows the trail of GARMIN GPS logs, and markers represent check-ins of foursquare in several places of interests. When a user moves a mouse over the marker, the name of the place and comment are shown.

Using the menu in the right side, the user can change the term and resolution of displayed logs. *DataSources* section under that has buttons which can switch on/off the appearance of each DataSource in the map.

C. Event Venues Log Map

Another example is a MashMap showing past conference venues, shown in Figure 6. This map shows the past venues of IEICE technical meetings (LOIS, OIS, OFS). This MashMap is associated with logs of venues of the meetings held since May, 1999 until March, 2012. A marker represents a venue of a meeting. With a mouse-over operation, the name of the place and the date of the meeting are shown.

V. RELATED WORK

There have been various research projects for map-based visualization of information with location. Kato et al. [9]

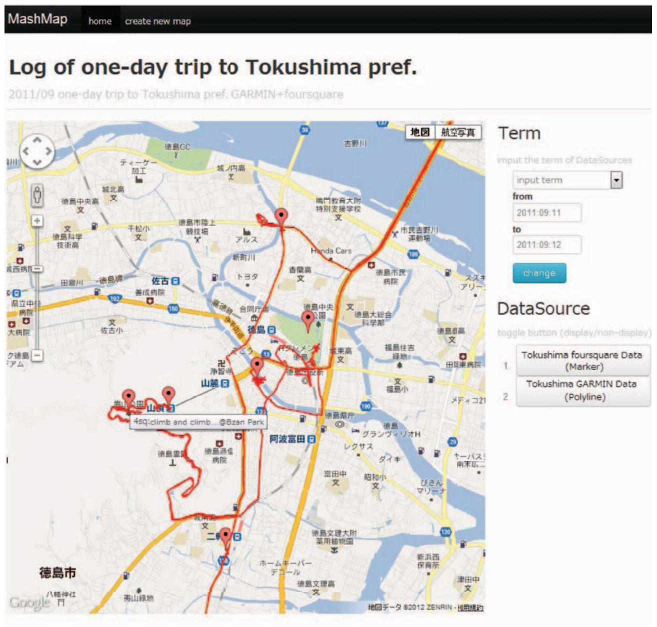

MashMap
home
create new map
Log of one-day trip to Tokushima pref.
2011/09 one-day trip to Tokushima pref. GARMIN+foursquare
Term
input the term of DataSources
input term
from
2011:09:11
to
2011:09:12
change
DataSource
toggle button (display/non-display)
1 Tokushima foursquare Data (Marker)
2 Tokushima GARMIN Data (Polyline)
地図
航空写真
徳島市

Figure 5. Log of one-day trip to Tokushima pref.

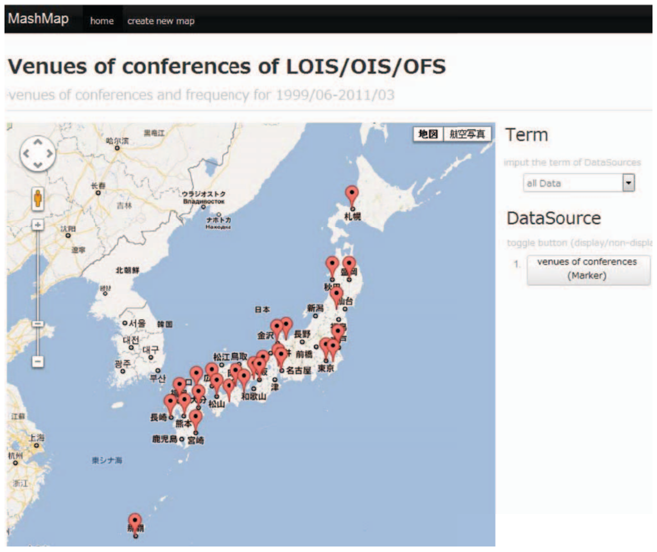

MashMap
home
create new map
Venues of conferences of LOIS/OIS/OFS
venues of conferences and frequency for 1999/06-2011/03
Term
input the term of DataSources
all Data
DataSource
toggle button (display/non-display)
1 venues of conferences (Marker)
地図
航空写真

Figure 6. Venues map of conferences (OFS/OIS/LOIS)

presented a system, called GDMS, which integrates several information for disaster prevention. Kuwabara et al. [10] developed a system, which associates time-series information with geometric location, allowing users to understand their relation. Tanaka et al. [11] is studying the mashup of various agricultural data on a map. The data include plant growth forecasting, disease forecasting, cultivate management, etc.

These existing researches respectively implemented their own custom maps by themselves. The custom map is created based on particular display format and proprietary data. Compared to them, the proposed MashMap framework is a general framework, which are not tightly coupled with specific applications or services. Using the proposed method, one can develop such custom maps efficiently and reliably.

VI. Conclusion

In this paper, we have proposed the *MashMap framework*, an application framework for visualizing lifelog with location onto a map. The proposed framework would contribute to rapid and reliable development of the location-based lifelog services. Our future work includes enhancement of MashMap framework. We plan to implement a feature that aggregates lifelog *without* location, and a feature that supports statistic and geometric analysis of lifelog data.

Acknowledgments

This research was partially supported by the Japan Ministry of Education, Science, Sports, and Culture [Grant-in-Aid for Scientific Research (C) (No.24500079), Scientific Research (B) (No.23300009)], and Kansai Research Foundation for technology promotion.

References

[1] K. Takata, J. Ma, B. O. Apduhan, R. Huang, and Q. Jin, "Modeling and analyzing individual's daily activities using lifelog," in *Embedded Software and Systems, 2008. ICESS'08. International Conference on.* IEEE, 2008, pp. 503–510.

[2] foursquare, http://foursquare.com/.

[3] S. Matsumoto, A. Shimojo, S. Kamada, and M. Nakamura, "Common data model and mashup api for integrating heterogeneouslifelogs," *IEICE Trans. on Information and Systems(Japanese Eds.)*, vol. J95-D, no. 4, pp. 758–768, April 2012.

[4] Jersey, http://jersey.java.net/.

[5] Weather Report Ch., http://weathernews.jp.

[6] Radiation Dose Data Gathering Map, http://minnade-map.net/.

[7] A. Shimojo, S. Matsumoto, and M. Nakamura, "Implementing and evaluating life-log mashup platform using rdb and web services," in *The 13th International Conference on Information Integration and Web-based Applications & Services (iiWAS2011)*, December 2011, pp. 503–506.

[8] Google Maps API, https://developers.google.com/maps/.

[9] T. Kato, M. Kobayashi, N. Sato, and T. Yotsuyanagi, "Prototype development of "geospatial disaster management mashup system"," *SEISAN KENKYU(Japanese Eds.)*, vol. 62, no. 4, pp. 377–380, 2010.

[10] K. Kuwahara, A. Hattori, and H. Hayami, "Development of time series geolocation information display system by map api and timeline api," *IPSJ SIG Notes(Japanese Eds.)*, vol. 2009, no. 33, pp. 109–114, 2009-03-11.

[11] K. Tanaka and M. Hirafuji, "Map interfaces using web map services in an agricultural model," *Agricultural Information Research(Japanese Eds.)*, vol. 18, no. 2, pp. 98–109, 2009.