On Integrating Heterogeneous Lifelog Services

Akira SHIMOJO, Saori KAMADA, Shinsuke MATSUMOTO, Masahide NAKAMURA
Graduate School of System Informatics, Kobe University
1-1, Rokkodai-cho, Nada, Kobe, Hyogo 657-8501, Japan
{shimojo, kamada}@ws.cs.kobe-u.ac.jp, {shinsuke, masa-n}@cs.kobe-u.ac.jp

ABSTRACT

This paper presents a framework that integrates different kinds of lifelog services. For efficient data mashup, we first propose the *lifelog common data model (LLCDM)*, which normalizes data structures and formats of heterogeneous lifelog records. We derive application-neutral data items by an interrogative analysis of what, why, when, who, where and how. We then implement the *lifelog mashup API (LLAPI)* to achieve standardized access to heterogeneous lifelogs. A case study of integrating practical lifelog services (Twitter, Flickr and GARMIN Connect) demonstrates the effectiveness of the proposed framework. It was shown that the development effort with the proposed APIs was reduced to 11.9%, compared to the conventional mashup development with the proprietary APIs.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-Based services*; D.2.12 [**Software Engineering**]: Interoperability—*Data Mapping*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed systems*

General Terms

Design, Experimentation

Keywords

lifelog, data integration, common data model, mashup API, Web services

1. INTRODUCTION

Lifelog (also known as *life caching*) is a social act to record and share human life events in an open and public form [14]. It is expected to be one of most promising applications in the ubiquitous society. The long-term logging would help a user to recall and improve daily life, as well as to recognize behaviors the user was never aware of. It might also be used for decision making and personal service recommendation.

Due to recent progress on storage and network technologies, a variety of *lifelog services* appear in the Internet. Using these services, we can easily store, publish and share various types of lifelogs on the Web. Popular lifelog services include; blog for writing diary, *Twitter* [15] for delivering one's status known as tweets, *Flickr* [16] for storing and sharing pictures, *FoodLog* [5] for photographing daily meals, *GARMIN Connect* [8] for managing training data with GPS, *karadalog* [11] for recording body measurements (weight, fat, etc.) for health and diet, and *Nemulog* [3] for logging sleeping hours.

In general, both storage and application are enclosed within the same lifelog service. Thus, various types of lifelog records are scattered over different service providers. Integrating such scattered lifelogs would implement more sophisticated and value-added services, rather than using them separately. We call *lifelog mashup* to represent such integration of different lifelog services over the Internet. As means of the lifelog mashup, some of lifelog services are providing Web services, APIs, blog components, etc., which allow external programs to access the lifelog data. However, the access methods are all different from service to service, since there is no standard specification. Such heterogeneity poses significant effort in developing mashup applications.

To support efficient lifelog mashup, this paper presents the *lifelog common data model (LLCDM)* and the *lifelog mashup API (LLAPI)*. The LLCDM prescribes a generic data schema for lifelog records, which does not rely on any specific lifelog service. To build an application-neutral model, we conduct an interrogative analysis, deriving data items from viewpoints of what, why, when, who, where and how. We then develop the LLAPI to manipulate data records conforming to the LLCDM. The LLAPI provide a standard way to retrieve data from heterogeneous lifelog services.

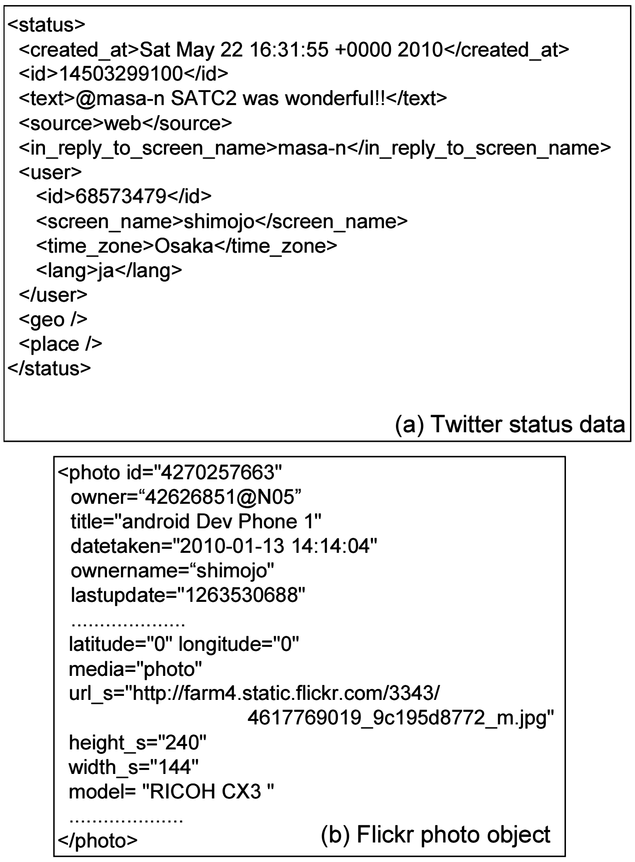
We also conduct an experiment of developing a practical lifelog mashup, called *LifeLogMaps*, with and without the proposed LLCDM/LLAPI. Integrating Twitter, Flickr and GARMIN Connect, the LifeLogMaps overlays tweets and pictures taken during the walking exercise on the path trail recorded. We evaluate the proposed method from viewpoints of development effort and execution performance.

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, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
iiWAS2010, 8-10 November, 2010, Paris, France.
Copyright 2010 ACM 978-1-4503-0421-4/10/11 ...$10.00.

2. PRELIMINARIES

2.1 Lifelog Services and Mashups

As introduced in Section 1, there are a variety of lifelog services available in the Internet. The great success of these



```
<status>
  <created_at>Sat May 22 16:31:55 +0000 2010</created_at>
  <id>14503299100</id>
  <text>@masa-n SATC2 was wonderful!!</text>
  <source>web</source>
  <in_reply_to_screen_name>masa-n</in_reply_to_screen_name>
  <user>
    <id>68573479</id>
    <screen_name>shimojo</screen_name>
    <time_zone>Osaka</time_zone>
    <lang>ja</lang>
  </user>
  <geo />
  <place />
</status>
```

(a) Twitter status data

```
<photo id="4270257663"
  owner="42626851@N05"
  title="android Dev Phone 1"
  datetaken="2010-01-13 14:14:04"
  ownername="shimojo"
  lastupdate="1263530688"
  ....................
  latitude="0" longitude="0"
  media="photo"
  url_s="http://farm4.static.flickr.com/3343/
                    4617769019_9c195d8772_m.jpg"
  height_s="240"
  width_s="144"
  model= "RICOH CX3 "
  ....................
</photo>
```

(b) Flickr photo object

Figure 1: Records of different lifelog services

services lies not only in information technologies, but also in the nature of human beings, loving to collect and store possessions, memories, experiences [14].

For these lifelog services, "what to log" is the selling point, and it varies from service to service. End users choose favorite services depending on their purpose. In general, storage and application are enclosed within the same lifelog service. Thus, various types of lifelog records are scattered over different service providers.

Integrating such scattered lifelogs may create more values rather than using them separately. In this paper, we define a term *lifelog mashup* to refer to such integration of different lifelogs to create a value-added service. For example, integrating "Twitter" and "Flickr", we may easily create a photo album with comments (as tweets). Also, integrating "FoodLog" and "karadalog", a doctor may find a symptom of lifestyle related disease by analyzing statistics on eating habits and body measurements.

2.2 Challenges in Lifelog Mashups

Assuming the lifelog mashup, some of the lifelog services are providing Web services, APIs, and/or blog components, which allow external programs to access the lifelog data. However, there is no standard specification among such APIs or data formats of the lifelog, which is as shown in the following example.

Figure 1 (a) shows a data record retrieved from Twitter, representing that a user "shimojo" made a tweet "SATC2 was wonderful!!" to a particular user "masa-n" on May 22, 2010. Figure 1 (b) is a data record of Flickr, describing information of a picture taken by a user "shimojo" on 2010-01-13. We can see that data schema of the two records are completely different. Although both are in the XML, there is no compatibility. Note also that these records were retrieved by different ways by using proprietary APIs.

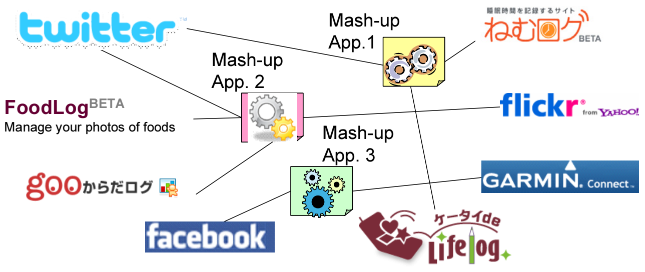


Figure 2: Conventional approach of lifelog mashup

There has been no other way but using the proprietary APIs for creating lifelog mashups. The conventional style of the lifelog mashup can be described as in Figure 2. We can see, in the figure, that each mashup application is *tightly coupled* with lifelog services (via proprietary APIs). Even for making the same query, the application has to use different methods for different services. Also different integration logic is needed for different combination of services. Thus, the conventional lifelog mashup tends to require great development effort. Also, the tight coupling declines portability and reliability of the application against the updates of the lifelog services.

2.3 Research Goal and Approach

To overcome the limitations of the conventional mashup, we design, implement and evaluate the *lifelog common data model (LLCDM)* and the *lifelog mashup API (LLAPI)* in this paper. Figure 3 shows the proposed method of the lifelog mashup. The data stored in heterogeneous lifelog services are transformed and aggregated in the LLCDM, which is an application-neutral form among the lifelog services. On top of the LLCDM, we define the LLAPI as generic interface to access the data, with which mashup applications are developed. The introduction of the LLCDM and the LLAPI achieves *loose coupling* between the application and individual lifelog services, and horizontal access to heterogeneous

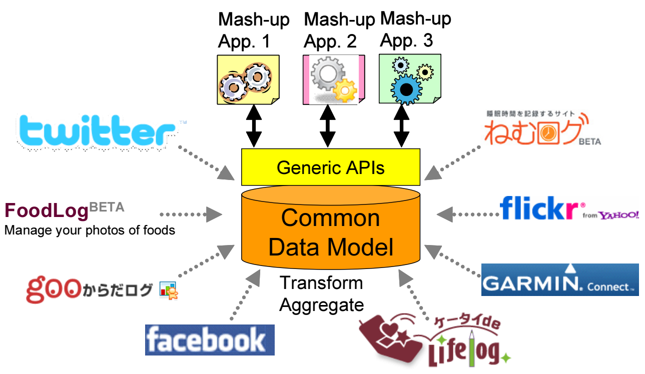


Figure 3: Proposed approach of lifelog mashup

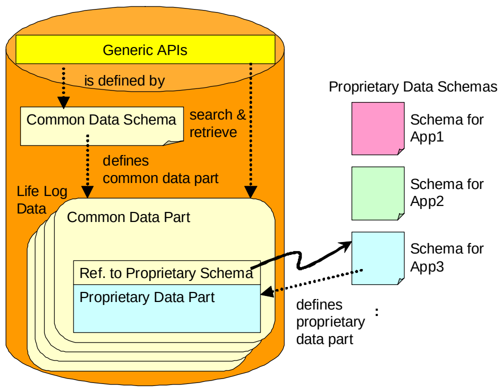


Figure 4: Architecture of lifelog common data model

lifelog data. Thus, the proposed approach will cope with the limitations in Section 2.2.

3. LIFELOG COMMON DATA MODEL

3.1 Overview

Figure 4 shows the architecture of the proposed LLCDM. The pillar represents a repository (i.e., database), which stores lifelog records (appear as rounded boxes) conforming to the LLCDM. Each record consists of "common" (i.e., application-neutral) data items and "proprietary" (i.e., appli cation-specific) data items.

The common data items are defined and interpreted by the generic *common data schema* within the LLCDM. Using the generic schema, the LLAPI implements application-neutral methods for operating lifelog data. On the other hand, the proprietary data items are not interpreted within the LLCDM. Instead, we delegate the data definition to external *proprietary schemas*, which are supposed to be prepared by individual lifelog services. For each record, the LLCDM just attaches a reference to the proprietary schemas.

Using this architecture, one can delegate all the data items to the proprietary schemas. This case is, however, equivalent to the one with the conventional proprietary APIs, and thus there is no merit using LLCDM.

The key issue for constructing a useful model is to derive as many common data items as possible. To achieve this, we conduct an interrogative analysis, investigating data items from the perspectives of what, why, who, when, where and how. In the following sections, `<tag>` represents a name of any data item.

3.2 Data items from WHAT Perspective

"What to log" varies from service to service, as discussed in Section 2.1. For example, Twitter records tweets (tagged by `<text>`), while Flickr stores pictures (`<photo>`). Thus, data items corresponding to WHAT perspective are quite application-specific. So, our LLCDM just stores the original data as it is, without interpreting them. The data is tagged by `<content>`, and its interpretation is delegated by an external schema pointed by the reference `<ref_schema>`.

For a given original lifelog record, there are design choices what data should be stored in `<content>` of the LLCDM. Our choice is to store the whole original record within `<content>` in order to prevent information loss.

3.3 Data items from WHY Perspective

For a given lifelog record, it is quite difficult for third person to know exactly "why the log is taken", unless explicitly described. From the WHY perspective, several data items can be considered, including `<purpose>`, `<reason>`, `<motivation>`, etc. However, some lifelog services can explain the purpose (e.g., recording weights and calories are for a diet). Also, most lifelog records are taken just for fun without strong reason. The reason or purpose can be derived by sophisticated data mining techniques. So, we consider that any data items w.r.t. WHY perspective are application-specific and outcomes of data mining. Thus, we exclude them from the LLCDM.

3.4 Data items from WHEN Perspective

"When the lifelog is taken" is one of the most application-neutral attribute. Not only the lifelog, but any kinds of logs usually record date (`<date>`) and time (`<time>`) to capture when the event occurred.

Strictly speaking, there might be several definitions for `<date>` and `<time>`. However, in the LLCDM we define them as "the date and time when the event occurred". As for the data format, we adopt the "Universal Time, Coordinated" (UTC) to maintain neutrality of the data. For the lifelog services not adopting the UTC format, we need to convert the original record in the LLCDM.

```
<date>   Date  YYYY-MM-DD (e.g., 2010-03-05)
<time>   Time  hh:mm:ss   (e.g., 12:34:56)
```

3.5 Data items from WHO Perspective

The lifelog is basically records of the human beings. So "Who caused the event" is an application-neutral attribute that must be associated with every lifelog record. In the LLCDM, we introduce a data item `<user>` to represent the subjective user that caused the event. Moreover, for events performed by multiple users, we propose to include `<party>` to say "With whom the event was caused", as well as `<object>` to describe "For whom the event was done". These are generic and useful data for representing relationships among multiple users.

As for the data format of `<user>`, `<object>` and `<party>`, we use "username" (i.e., account name) used by individual lifelog services. The username is a mandatory attribute for lifelog services, so it is always available. Also, the username is the a primary identity of each user. So, we decided to use it within the LLCDM.

```
<user>     String username   (e.g., shimojo)
<party>    String username   (e.g., saorin)
<object>   String username   (e.g., masa-n)
```

3.6 Data items from WHERE Perspective

"Where the lifelog is taken" is also an application-neutral attribute. For instance, the GPS coordinates directly represent where I am (was). A picture may have information on where it was taken. Height and weight may be associate with a place where they were measured. So, the information of place, say `<location>`, can be easily associated with any kinds of data. Hence, we include `<location>` in the LLCDM.

As for the data format representing `<location>`, we use coordinates (latitude and longitude) as well as the street address. The coordinates pin-points the exact location, while the street address is intuitive for human users to query the

Table 1: Common data schema of LLCDM

Perspective	Data Tag	Data Format	Description
WHAT	<content>	Binary original_data	Contents of the log (whole original data)
	<ref_schema>	URL url	URL references to external schema
WHY	n/a	n/a	n/a
WHEN	<date>	Date YYYY-MM-DD	Date when the log is created (in UTC)
	<time>	Time hh:mm:ss	Time when the log is created (in UTC)
WHO	<user>	String username	Subjective user of the log
	<party>	String username	Party involved in the log
	<object>	String username	Objective user of the log
WHERE	<address>	String street_address	Street address where the log is created
	<latitude>	Double latitude	Latitude where the log is created
	<longitude>	Double longitude	Longitude where the log is created
HOW	<application>	String application	Service/application by which the log is created
	<device>	String device	Device with which the log is created

data. Consequenely, we deceded to use all of `<latitude>`, `<longitude>` and `<address>` as subnodes of `<location>`.

```
<location>
  <latitude>  double latitude  (e.g., 34.72631)
  <longitude> double longitude (e.g., 135.235321)
  <address> String address (e.g., 1-1 Kobe St...)
</location>
```

3.7 Data items from HOW Perspective

Preferably, "how the lifelog is recorded" should not depend on specific lifelog services. For example, Twitter is not only the way to record a tweet (i.e., short text). Moreover, the same tweet can be produced from several devices (e.g., cell-phone, PDA, PC, etc.) Thus, basically, means and devices used for the lifelog should be independent of contents of the lifelog. In fact, however, there exist a dependency between a lifelog service and the way of lifelog recording, since supported devices and methods are usually limited by the service. Based on this, from HOW perspective we decided to include `<application>` and `<device>` as the application-neutral data items in the LLCDM. `<application>` represents the name of application used for the lifelog recording. `<device>` is the name of device used. These data items are attached to every lifelog record.

```
<application> String app_name (e.g., Flickr)
<device>      String dev_name (e.g., RICOH CX3)
```

Table 1 summarizes all data items of the LLCDM, derived from the interrogative analysis.

3.8 Transforming Lifelog to LLCDM

Our next interest is how to transform the original lifelog record to fit to the LLCDM. We present a transformation method consisting of the following three steps.

(Step 1) Defining Data Mapping

This step defines a *data mapping* from every data item in the original record to the one in the LLCDM. In general, the data schema in the original record varies from service to service. Hence, the mapping should be defined for every lifelog service to integrate.

(Step 2) Converting Data Format

Next, we convert the format of the original data value to the one of the LLCDM. The conversion should be carefully done not to change the semantics.

```
<lifelog id=10000002>
  <date>2010-05-22</date>
  <time>16:31:55</time>
  <user>shimojo</user>
  <party/>
  <object>masa-n</object>
  <location>
    <latituade/>
    <longitude/>
    <address/>
  </location>
  <application>twitter</application>
  <device>web</device>
  <ref_schema>http://apiwiki.twitter.com/
  </ref_schema>
  <content> <status><created_at>.....
    <text>@masa-n SATC2 was wonderful!!</text>
    .....</status>
  </content>
</lifelog>                                   (a) Twitter data
```

```
<lifelog id=20000015>
  <date>2010-01-13</date>
  <time>14:14:04</time>
  <user>shimojo</user>
  <party/>
  <object/>
  <location>
    <latituade/>
    <longitude/>
    <address/>
  </location>
  <application>flickr</application>
  <device>RICOH CX3</device>
  <ref_schema>http://www.flickr.com/services/api/
  </ref_schema>
  <content> <photo id="4270257663"
    ....................
    title="android Dev Phone 1"
    url_s="http://farm4.static.flickr.com/3343/
               4617769019_9c195d8772_m.jpg"
    ....................></photo>
  </content>
</lifelog>                                   (b) Flickr data
```

Figure 5: Lifelog records converted to LLCDM

(Step 3) Preserving Original Data

Finally, we copy the whole original record to `<content>`. Then, we obtain a URL of the schema of the record from the service provider, and put the URL in `<ref_scheme>`. This prevents the information loss during the data transformation, and maintains the compatibility with the conventional libraries and/or APIs provided by the lifelog services.

(Example) Converting Twitter Record to LLCDM

As an illustrative example, we here transform the data record of Twitter in Figure 1(a). In Step 1, we define the data mapping. The date information in `<created_at>` is divided and mapped to `<date>` and `<time>`. The username shown in `<screen_name>` is mapped to `<user>`. Also, the username appearing after "@" in`<text>` represents a user to be replied, so is mapped to `<party>`. The source device in `<source>` is mapped to `<device>`. The name of the service "Twitter" is mapped to `<application>`.

In Step 2, we convert the format. For example, the date "Sat May 22 16:31:55 +0000 2010" is divided and transformed into "2010-05-22" and "16:31:55". In Step 3, we copy the whole data of `<status>` into `<content>` as it is. Then, we specify the URL "`http://apiwiki.twitter.com/`" of the Twitter-API specification in `<ref_scheme>`. The resultant record is shown in Figure 5(a), which conforms to the LL-

CDM. Similarly, the data record of Flickr (see Figure 1) can be transformed into the one in Figure 5(b).

4. LIFELOG MASH UP API (LLAPI)

This section presents APIs for searching and retrieving lifelog data conforming to the LLCDM, which can be used extensively for building lifelog mashups.

4.1 getLifeLog()

This API searches and retrieves lifelog records matching a given query. The query is specified by the data items of the LLCDM, i.e., `<date>`, `<time>`, `<party>`, `<object>`, `<location>`, `<application>` and `<device>`. It also takes a "select" parameter to select particular data items from the resulting records (like SELECT statement of SQL).

Interface:

```
LifeLog[] getLifeLog(date, time, user, party,
object, location, application, select);
```

Parameters:

```
date:         Query of <date>
time:         Query of <time>
user:         Query of <user>
party:        Query of <party>
object:       Query of <object>
location:     Query of <location>
application:  Query of <application>
select:       List of items to be selected
```

Query:
An expression consisting of literals (”Quoted String”), logical OR ('+'), and wild-card (*).

Return Value:
A list (array) of lifelog data records matching the given query. Each record can be represented by a hashmap where keys are data tags in Table 1.

Example:
Retrieve lifelog records of the user "shimojo.akira" that are taken on March 1st, 2010 with "Twitter" or "Flickr". From the result, select data items of date, time, username, Twitter's tweet (`<text>`), Flickr's URL of a picture (`<url_s>`) and the title of the picture (`<title>`).

```
getLifeLog({date        => "2010-03-01",
            user        => "shimojo.akira",
            application => "Twitter+Flickr",
            select      => "date+time+user+text
                                  +url_s+title"})
```

4.2 getLifeLogNearestTime()

The API `getLifelog()` is quite generic to get all lifelog records matching the given query. However, one may want to get the *single nearest matching*. We are currently supporting an API for searching the nearest time record as shown below. In the future, we plan to consider other criteria such as the nearest location, the nearest user, etc.
Interface:

```
LifeLog getLifeLogNearestTime(date, time, user,
  application);
```

Parameters:

```
date:         Query of <date>
time:         Query of <time>
user:         Query of <user>
application:  Query of <application>
```

Return Value:
A lifelog record taken on the nearest date and time matching the given query.

Example:
Get a lifelog record of the user "shimojo.akira" with GARMIN Connect, whose recorded date is nearest to 12:34:56, March 1st, 2010.

```
getLifeLog({date        => "2010-03-01",
            time        => "12:34:56",
            user        => "shimojo.akira",
            application => "GARMIN"})
```

5. IMPLEMENTATION

5.1 LLCDM Data Converter and LLAPI

We have implemented a set of Perl scripts that convert lifelog data records into the LLCDM. Currently, data conversion of Twitter, Flickr and GARMIN Connect are supported. The scripts are implemented using open-source libraries such as XML::Simple, Net::Twitter, Flickr::API, and Time::Piece, and are comprised of 930 lines of code. For the data conversion, each script first obtains the original lifelog data using proprietary APIs provided by individual lifelog services. Then, it converts the data into the LLCDM based on the conversion method presented in Section 3.8. Finally, the converted data is stored in an XML format.

Based on Section 4, we have then implemented the LLAPI in the Perl language. The LLAPI is comprised of around 400 lines of code, and can be used as a perl library. The LLAPI can be also deployed as a REST Web service to increase programmatic interoperability.

5.2 Providing Proposed Method as Service

In providing the proposed LLCDM and LLAPI as public services (see Figures 3 and 4), there are two options: *offline mode* and *online mode*. The offline mode uses scheduled batch processes to (1) gather all available lifelog records from various services, (2) convert the records into the LLCDM, and (3) store the converted records in the repository. When a request comes through the LLAPI, the system searches and retrieves the already-converted records. The offline mode can provide light-weight access for the LLAPI, as the data conversion has been done. The drawback is that the latest data is not necessarily available in the repository.

On the other hand, the online mode performs the *on-the-fly* data gathering and conversion when the request arrives. The online mode can access the latest lifelog data and can save storage space. However, it poses huge overhead in the LLAPI. Since both modes have own advantage and drawback, they should be chosen according to characteristics of the mashup applications to be developed. Our current implementation supports the offline mode only. The online mode will be implemented in our future work.

6. EVALUATION

6.1 Building a Mashup Application

In this section, we conduct an experiment of developing a practical lifelog mashup application, called *LifeLogMaps*, by integrating three heterogeneous services: Twitter, Flickr and GARMIN Connect. To evaluate the effectiveness of the proposed method, the application is developed in two ways: one process with the conventional APIs, and another process with the proposed LLCDM and LLAPI.

LifeLogMaps

LifeLogMaps is an application supporting walking exercise, which overlays short text, pictures and walking trails (GPS) recorded during the exercise onto a single map of Google Maps. LifeLogMaps obtains the three kinds of data from Twitter, Flickr and GARMIN Connect, respectively, and integrates the data according to date and time.

A user takes for a walking exercise with a GARMIN GPS receiver, a digital camera and hand-held device (PDA, cell phone, etc). During the exercise, the user writes short text coming to mind using Twitter, and takes pictures wherever the user wants. After the exercise, the user uploads favorite pictures to Flickr, and also GPS data to GARMIN Connect. Next, LifeLogMaps automatically downloads these data from Twitter, Flickr and GARMIN Connect, integrates them, and outputs the integrated data in a JSON format. Finally, using Javascript the JSON data is passed to Google Maps API to generate the integrated map.

Figure 6 is a screenshot of LifeLogMaps, showing that a user "Saori" took for a walk to Takamatsu city, Japan, on January 9th, 2010. In the map, two kinds of markers appear along with her walking trail, indicating locations where a picture or short text is recorded. When a user clicks a marker, the data is displayed in a popup window.

Note that LifeLogMaps is implemented by reusing the *existing* services, without creating dedicated services. We can see that integrating three existing services provides more value-added contents, rather than using them separately.

Implementation with Conventional APIs

First, we explain the conventional process of implementing LifeLogMaps without the LLAPI. The process consists of the following four steps:

Step1 (Obtain original lifelog data) Obtain lifelog data using proprietary APIs or Web services provided by Twitter, Flickr and GARMIN Connect.

Step2 (Extract data items) Extract necessary data items by parsing data records obtained in Step1. Specifically, select [date, time, tweet] from Twitter, [date, time, title of picture, URL of picture] from Flickr, and [date, time, latitude, longitude] from GARMIN.

Step3 (Join data items) Join the data items extracted in Step2 on their date and time. Since the format and interpretation of date and time are different service to service, we convert them in a standard format before executing the join. As a result, a list of data records [date, time, latitude, longitude, tweet, title of picture, URL of picture] is generated. Finally, the list is output to a file in the JSON format.

Step4 (Create LifeLogMaps) Visualize the JSON data using Google Maps APIs. Adjunct coordinates (latitude, longitude) are linked by a line for representing a trail. For each tweet (or picture), a marker is placed on the corresponding coordinates. Finally, the marker is configured to display the contents in a pop-up window when clicked.

Implementation with Proposed Method

Next, we explain the implementation using the proposed LLCDM and LLAPI. The process becomes quite simple and systematic, since all of the three services and derived data can be accessed in a uniformed way.

Step1' (Obtain and join lifelog data) For a given date, time and username, obtain data records of Twitter by executing `getLifeLog()`. Let t be every Twitter record. For each date (`t->{date}`) and time (`t->{time}`), execute `getLifeLogNearestTime()` to retrieve the nearest GARMIN record g. Then join t and g. Do the same thing for every Flickr record. To these joined records, add all records of GARMIN retrieved by `getLifeLog()`. Finally, output the record to a file in a JSON format.

Step2' (Create LifeLogMaps) Same as Step4 of the conventional method, and thus omitted.

6.2 Experiment

In the experiment, two students of our laboratory formed a project, developing two versions of LifeLogMaps using the conventional and the proposed methods. The students had programming experience of C, Java, JavaScript, Perl, but no a-priori knowledge of APIs of the lifelog services. The objective of the experiment is to evaluate the proposed method from two aspects: (1) contribution to the effort and productivity in mashup development, and (2) contribution to the performance of the application.

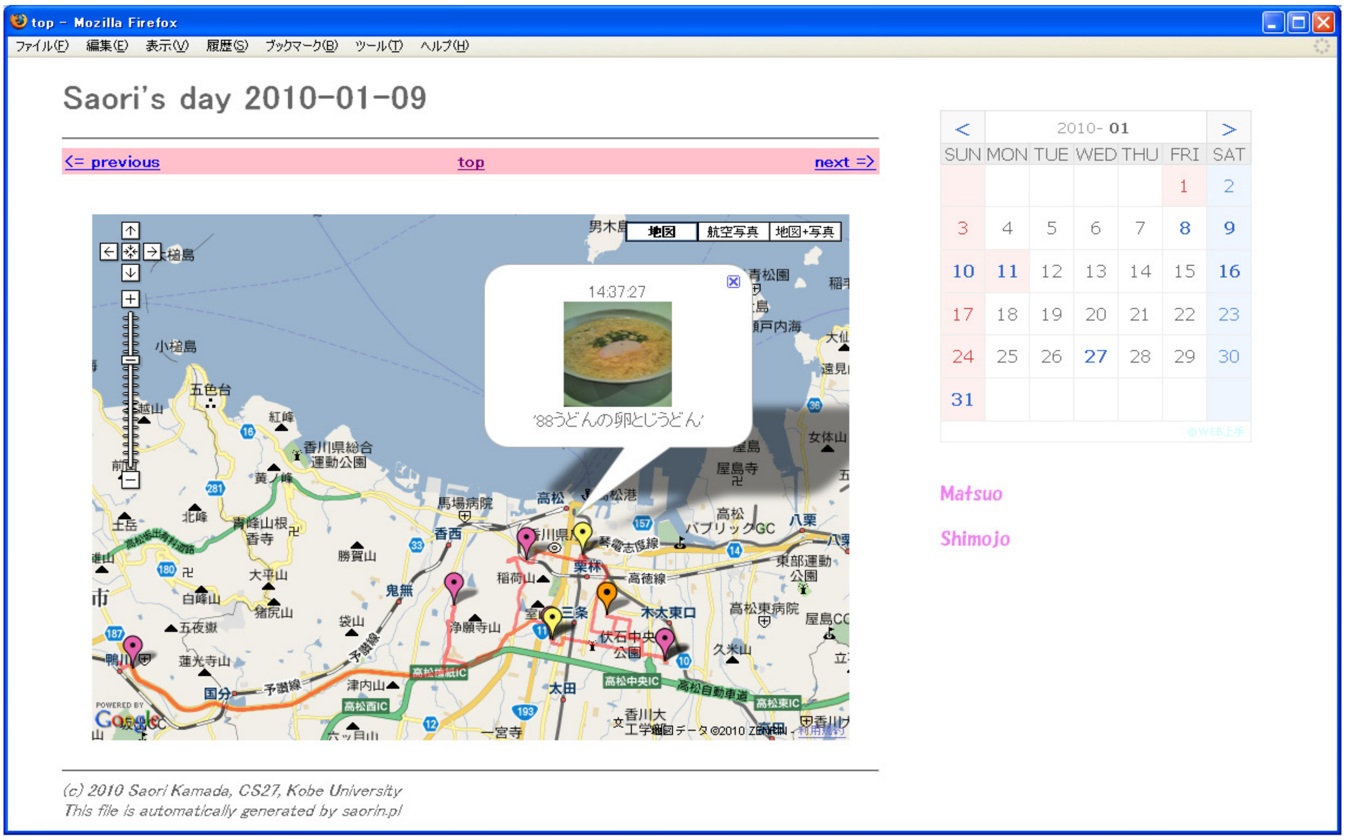
Effort and Productivity

For each development process, we measured the development effort in man-day, and the size of products w.r.t. the number of modules and the lines of code.

Table 2 shows the result. It can be seen in the table that the development with the proposed method achieves 9 times efficiency in the development effort and 5 times efficiency in the product size. After the experiment, we interviewed the students to validate the result.

The reasons why the conventional method required much effort are explained as follows. First, it was necessary in Step1 to learn the usage of proprietary APIs from different service providers, which was time-consuming. In Step2, for each of three services, the students had to understand the data structure of the lifelog record, and had to implement a parser module to extract necessary data items. In Step3, the students felt bothersome to understand and unify the different data formats. The effort needed for the above things must be accumulated if more and more services are integrated.

On the other hand, opinions for the proposed method were positive. Once having learned the usage of the LLCDM and LLAPI, the students could concentrate on implementing the mashup logic only, without spending effort for *syntactical* differences among services. This significantly reduced the code size as well as development effort. Also,



(c) 2010 Saori Kamada, CS27, Kobe University
This file is automatically generated by saorin.pl

Figure 6: A screenshot of LifeLogMaps

Table 2: Development effort and product size

Development method	Number of modules	The total lines of code	Development man-hour
Conventional	7	394 LOC	42 man-day
Proposed	2	82 LOC	5 man-day

Table 3: Application performance

Development method	Input Data Set							
	8-Jan	9-Jan	10-Jan	11-Jan	16-Jan	27-Jan	29-Jan	31-Jan
Conventional	3.9	6.4	4.0	3.8	4.2	5.0	4.6	3.9
Proposed	44.7	43.3	20.7	17.4	45.3	8.6	13.2	45.8
Size of Processed Log (KB)	2411	1635	2150	1028	1695	346	699	1307

even if the number of services increases, the effort for retrieving the lifelog data would be the same, as long as the service is supported by the proposed framework. Thus, the more lifelog services are integrated, the more efficiency the proposed method can achieve.

Application Performance

Next, we measured the execution time to compare the performance of the two versions of LifeLogMaps. We used the actual 8-day data of Twitter, Flickr and GARMIN Connect collected during January 2010. For each version of LifeLogMaps, we measured execution time taken for LifeLogMaps to generate the JSON data. The result is shown in Table 3. Each column represents the execution time in second for each data set. Just for the reference, we show the size of lifelog records processed in the last row.

We can see that LifeLogMaps with the proposed LLAPI took much more execution time (3 to 11 times worse) than the conventional. The reason of such low performance is explained as follows. In our current implementation, the LLCDM repository (see Figure 4) is just a file system containing converted data as XML files. Hence, for a given query the LLAPI has to search the files sequentially, yielding expensive overhead. Introducing an XML database and caching mechanism would improve the performance drastically, which is our future work.

6.3 Feasiblity to Other Lifelog Services

To evalute feasibility of the LLCDM, we have applied the LLCDM to 11 practical lifelog services (6 explained before, plus Life Pod [10], Life-X [13], LifeSpaceTime [6], LOGPI [12] and AcTrek [17]). For each service, we checked if the original data record can be converted to the LLCDM by the method in Section 3.8. Table 4 shows the result.

In the table, $\surd$ (or $-$) represents that the data item is originally supported (or unsupported, respectively) in the service. A comment describes that the data item is originally not supported, but can be produced by other data source.

It can be seen that for every service there exist at least 3 items as the common data of LLCDM. Thus, we believe that the proposed LLCDM is generic and neutral enough to accomodate data from heterogeneous lifelog services.

6.4 Related Work

Several studies have been reported on the mashup of Web services. For example, Lizcano et al. [7] and Lorenzo et al.

Table 4: Feasibility of LLCDM to various lifelog services

	Twitter [15]	Flicker*[16]	FoodLog[5]	GARMIN Connect[8]	karadalog[11]	nemu log [3]	Life-Pod[10]	Life-X[13]	LifeSpaceTime[6]	LOGPI[12]	AcTrec[17]
<content>	tweets	pictures	food (photo)	GPS, heartbeat, cadence	body measurements	sleeping log	user's actions	miscellaneous, aggregation	miscellaneous	tweets	action history
<date>	√	√	√	√	√	√	√	√	√	√	√
<time>	√	√	√	√	-	-	√	√	√	√	√
<user>	√	√	√	√	√	√	√	√	√	√	√
<party>	#Topics	-	-	-	-	-	life:with	-	-	-	-
<object>	√	-	-	-	-	-	-	-	To addressed log	reply log	-
<location>	√	√	-	√	-	-	√	√	location log	-	√
<application>	√	√	√	√	√	√	√	√	√	√	√
<device>	<source>	taken with	-	<name>	-	-	prf:Model	-	-	-	-

[4] presented frameworks supporting end-users for integrating Web services. Maximilien et al. [9] proposed an online service to share and reuse mashup components on the Web. Abiteboul [1] presented a notion of mashlet to provide necessary information of components for global mashup. Basically, these methods support just *operation integration* in the conventional mashup. The proposed method achieved also *data integration* with the LLCDM by focusing our domain within lifelog services.

Amato et al. [2] proposed a common data model for sensor network systems, in order to unify proprietary data formats provided by multi-vendor sensor devices. The data model (schema) is designed specifically for sensors, simply consisting of sensor characteristic and its measurement. Hence, it cannot be applied directly to the lifelog services which tend to have more attributes and complex structures.

7. CONCLUSION

In this paper, we have proposed the lifelog common data model (LLCDM) and the lifelog mashup API (LLAPI) for efficient integration of heterogeneous lifelog services. The proposed method was evaluated through an experiment, where a practical mashup application, LifeLogMaps, was developed. It was shown that the proposed method significantly improved the efficiency of the mashup development. Our future work includes the performance improvement of the LLAPI as well as supports for more services. It is also interesting to consider the privacy and copyright issues for business and commercial use.

8. ACKNOWLEDGMENTS

This research was partially supported by the Japan Ministry of Education, Science, Sports, and Culture, Grant-in-Aid for Young Scientists (B) (No.21700077).

9. REFERENCES

[1] S. Abiteboul, O. Greenshpan, and T. Milo. Modeling the mashup space. In *WIDM '08: Proceeding of the 10th ACM workshop on Web information and data management*, pages 87–94, New York, NY, USA, 2008.

[2] F. Amato, V. Casola, A. Gaglione, and A. Mazzeo. A common data model for sensor network integration. *Complex, Intelligent and Software Intensive Systems, International Conference*, 0:1081–1086, 2010.

[3] BOstudio, Inc. nemulog. `http://www.nemulog.jp/`.

[4] G. Di Lorenzo, H. Hacid, H.-y. Paik, and B. Benatallah. Data integration in mashups. *SIGMOD Rec.*, 38(1):59–66, 2009.

[5] K. Kitamura, T. Yamasaki, and K. Aizawa. Foodlog: Capture, analysis and retrieval of personal food images via web. In *CEA '09: Proceedings of the ACM multimedia 2009 workshop on Multimedia for cooking and eating activities*, pages 23–30, New York, NY, USA, 2009. ACM.

[6] LifeSpaceTime. LifeSpaceTime. `http://www.lifespacetime.com/`.

[7] D. Lizcano, J. Soriano, M. Reyes, and J. J. Hierro. Ezweb/fast: Reporting on a successful mashup-based solution for developing and deploying composite applications in the upcoming web of services. In *iiWAS '08: Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*, pages 15–24, New York, NY, USA, 2008. ACM.

[8] G. Ltd. Garmin. `http://connect.garmin.com/`.

[9] E. M. Maximilien, A. Ranabahu, and K. Gomadam. An online platform for web apis and service mashups. *IEEE Internet Computing*, 12:32–43, 2008.

[10] A. Minamikawa, N. Kotsuka, M. Honjo, D. Morikawa, S. Nishiyama, and M. Ohashi. Rfid supplement for mobile-based life log system. *Applications and the Internet Workshops, IEEE/IPSJ International Symposium on*, 0:50, 2007.

[11] NTT Resonant Inc. karadalog. `http://karada.goo.ne.jp/`.

[12] paperboy&co. . Logpi. `http://logpi.jp/`.

[13] Sony Marketing (Japan) Inc. Life-x. `http://life-x.jp/`.

[14] Trend Watching .com. Life caching – an emerging consumer trend and related new business ideas. `http://trendwatching.com/trends/LIFE_CACHING.htm`.

[15] Twitter, Inc. twitter. `http://twitter.com/`.

[16] Yahoo. Flickr. `http://www.flickr.com/`.

[17] H. Yamane and K. Nagao. Actrec: Personal action support by situation awareness and recording [in japanese]. *The 66th National Convention of IPSJ*, 66(3):115–116, 2004.