Implementing and Evaluating Life-Log Mashup Platform Using RDB and Web Services

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

ABSTRACT

In order to support efficient integration of heterogeneous lifelog services, we have previously proposed a *lifelog mashup platform* consisting of the *lifelog common data model (LLCDM)* and the *lifelog mashup API (LLAPI)* to access the standardized data. However, it had the performance bottleneck, and was poor in the portability. To cope with these problems, we re-engineer the LLCDM and the LLAPI with the relational database MySQL and the Web services, respectively. Furthermore, we evaluate the practical feasibility through an actual development project.

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, Performance

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 [1, 2, 4]. Due to recent progress, a variety of *lifelog services* appear in the Internet. Popular lifelog services include; blog for writing diary, *Twitter* for delivering tweets, *Flickr* for storing pictures, *BodyLogService* for recording body measurements like weight and fat.

In general, both storage and application are enclosed within the same lifelog service. 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.

To support efficient lifelog mashup, we presented a *lifelog mashup platform* consisting of the *lifelog common data model (LLCDM)* and the *lifelog mashup API (LLAPI)* in the previous work [3].

However, there were two major limitations in the previous implementation. The LLAPI was implemented as *Perl libraries* that access the local file system storing XML data of the LLCDM. Therefore, the API had quite *low performance* and *poor portability*. Also, the library-based development posed developers *artificial dependencies* in programming language, platform and the version maintenance.

In this paper, we re-engineer the LLCDM and LLAPI using a relational database and Web services, respectively. Another contribution of this paper is to evaluate the *practical feasibility* of the LLCDM and the LLAPI. Specifically, we conduct an experiment where developers implement a practical mashup application, called *Tabetalog*, by integrating two heterogeneous services: Flickr and BodyLogService. The application is developed by two processes: one with the conventional APIs, and another with the proposed LLCDM and LLAPI. We compare the two different approaches from viewpoints of product and process.

2. PRELIMINARIES

2.1 Lifelog Services and Mashups

As mentioned in Section 1, there are a variety of lifelog services available in the Internet and integrating different 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).

Expecting the lifelog mashup, some lifelog services are already providing APIs and/or blog components to access the lifelog data. However, there is no standard specification among such APIs or data formats of the lifelog.

To support efficient lifelog mashup, we have previously proposed a *lifelog mashup platform* [3]. The platform consists of the *lifelog common data model (LLCDM)* and the *lifelog mashup API (LLAPI)*, as shown in Figure 1. The data stored in heterogeneous lifelog services are transformed and aggregated in the LLCDM, which is an application-neutral

iiWAS2011, 5-7 December, 2011, Ho Chi Minh City, Vietnam.

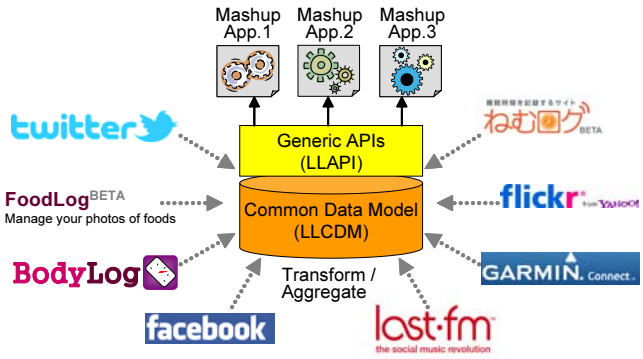

Figure 1: Proposed lifelog mashup platform [3]

form among the lifelog services. The LifeLog Mashup API (*LLAPI*) is for searching and retrieving lifelog data conforming to the LLCDM. The following shows an API that returns lifelog data matching a given query.

```
getLifeLog(date, time, user, party,
           object, location, application, device)
```

Using `getLifeLog()`, heterogeneous lifelogs can be accessed uniformly without proprietary knowledge of lifelog services.

2.2 Limitations in Previous Work

In [3], we have prototyped the LLCDM repository and the LLAPI. However, we found in the prototype that there were two major limitations for practical use.

The first limitation is in the *performance*. In the prototype, the LLCDM repository was just a file system containing converted data as raw XML files. Thus, we could not ignore the overhead if we use the LLAPI for the *online* data mashup.

Another limitation is in the *portability*. In the prototype, we provided the LLAPI as a program library written in the Perl language. Therefore, there was no choice for developers to use other languages for building mashup applications.

To overcome these limitations, we aim to achieve the following two goals.

G1: Improve performance and portability of the prototype.

G2: Evaluate the practical feasibility of the LLAPI.

To achieve the goal G1, we put all the lifelog data in a *relational database*, instead of having the data as raw XML files. We aim faster data search and access. We re-engineer the LLAPI as a wrapper program of the SQL and deploy the program as a Web service, providing the platform-independent access to the lifelog data.

To achieve the goal G2, we conduct an application development experiment. In the experiment, we ask subjects to implement two versions of a mashup application, with and without the proposed LLAPI and the LLCDM. The conventional and proposed approach will be compared from the viewpoints of process and product.

3. ENHANCING LIFELOG MASHUP PLATFORM

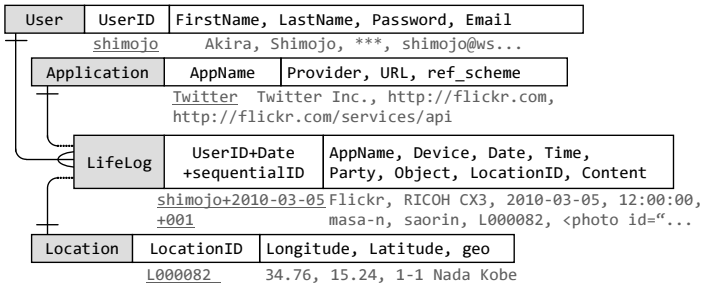

Figure 2: ER diagram for the LLCDM repository

3.1 RDB for managing LLCDM repository

To improve the performance, we introduce the relational database to manage the LLCDM repository (See Figure 1). Figure 2 depicts the proposed ER diagram of the LLCDM. A box represents an entity (i.e., table), consisting of an entity name, a primary key and attributes. We enumerate instances below each entity to support understanding. A line represents a relationship between entities, where $+\!\!-\!\!\in$ denotes a parent-children relationship, and $+\!\!-\!\!\cdots$ denotes a reference relationship. The diagram consists of the four tables.

User table manages the user information, consisting of UserID, Username, Password and E-mail address. In Application table, we manage the applications from which we retrieve the lifelog data record. The attributes of the table are Application name, Provider, URL, Description and Reference URL. LifeLog is a main table to store lifelog data retrieved, consisting of every item of the LLCDM. The primary key is a composite key of UserID, Date and serial number, since our empirical study shows that the lifelog data is often searched by UserID and Date. Location table stores the location data in the WHERE perspective. The table consists of address, latitude and longitude.

3.2 Importing Lifelog Data to LLCDM Repository

Next, we consider how to import data from heterogeneous lifelog services to the LLCDM repository. The process consists of the following three steps:

(Step1) Obtain original data: We obtain the original lifelog data from heterogeneous lifelog services and store the data in XML.

(Step2) Transform data to LLCDM: We then transform the original raw data to the LLCDM format. In this step, we maps the original data to each item of the LLCDM, based on a *conversion rule* defined for each individual service in [3].

(Step3) Insert data into database: Finally, we insert the XML into the database. We parses the converted XML data, extracts the attributes and inserts the values to appropriate tables in Figure 2.

The reason why we split the data import task into three steps is to improve the system maintainability. For example, even if the specification of Flickr API is changed, we just update the crawler for Flickr only, and we need not to touch other programs.

Table 1: Comparison of execution time

	Q1	Q2	Q3	Q4
SOAP(sec)	0.131	1.006	0.281	0.422
REST(sec)	0.015	0.100	0.019	0.025
OLD(sec)	4.238	4.028	4.254	0.581
# of items	36	119	195	449
data size (kB)	118	381	1,450	630

3.3 Re-engineering LLAPI

Since the lifelog data is imported in the relational database reflecting the LLCDM format, the LLAPI is easily implemented as a query program wrapping an SQL statement. We implemented the LLAPI in the Java language. We used iBatis OR-mapper for marshaling tuples into objects. The advantage of using the OR-mapper is to decouple the SQL statements from the Java code, which improves the robustness and the maintainability of the system.

Finally, we deploy the LLAPI as a Web service. A great advantage of using the Web service is to free the LLAPI from the artificial dependencies on the language and platform. We have deployed the LLAPI using the Apache Axis2 Web service framework. Now that the LLAPI can be accessed by both SOAP and REST Web service protocols.

3.4 Evaluating Performance

We compare the performance of the old prototype (see Section 2.2), and the new implementation developed in this paper. At the time of the evaluation, 1,591 records of lifelog data were stored in the MySQL database or in the XML files. For the performance measurement, we developed Perl clients to invoke the LLAPI with the following queries.

Query 1: Get lifelog data taken between 2010-10-15 and 2010-10-16

Query 2: Get lifelog data taken between 2010-09-01 and 2010-09-30

Query 3: Get lifelog data taken between 09:00:00 and 10:15:00 on any date.

Query 4: Get lifelog data taken by user "Shimojo"

For each query, we executed the LLAPI five times, and measured the average execution time. For the new version, we tried both SOAP and REST invocations. Just for the reference,

Table 1 shows the execution time, measured for the four queries. The fourth and fifth rows of the table respectively show the number of items and the data size retrieved by the queries. It can be seen, in Table 1, that the effective performance for the new version (REST) has been improved significantly from 23 to 275 times. Much of this improvement owes to the performance of MySQL. Also, we can see that REST performs 8 to 16 times better than SOAP in every case. However, we can still accept the SOAP overhead compared with the old version. Thus, we believe that the re-engineering of the mashup platform achieved the goal G1 in Section 2.2.

4. APPLICATION DEVELOPMENT EXPERIMENT WITH LLAPI

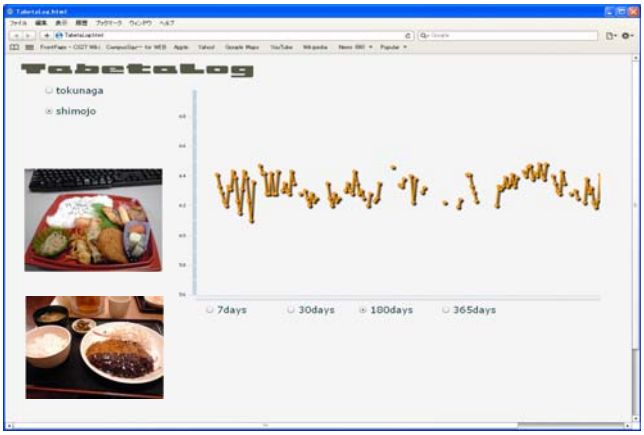

Figure 3: A screenshot of TabetaLog

4.1 Mashup Example: TabetaLog

This section conducts an experimental evaluation of developing a practical lifelog mashup application, called *TabetaLog*, in order to achieve the goal G2 in Section 2.2.

The *TabetaLog* is a mashup application supporting user's eating habits, visualizing weight and pictures of foods taken every day. This application obtains the picture data and the weight data from Flickr and BodyLogService respectively, and integrates the record according to the date. A screenshot of TabetaLog is shown in Figure 3. The process of implementing TabetaLog consists of the following four steps:

Step1 (Obtain original lifelog records): Obtain lifelog data using proprietary APIs of Flickr and BodyLogService or the proposed LLAPI.

Step2 (Extract data items): Extract necessary data items by parsing records obtained in Step1. Specifically, select [date, user, value] from BodyLogService, and [time, picture URL] from Flickr.

Step3 (Join data items): Join the data items extracted in Step2 on their date. Finally, the joined records are dumped to a file in JSON format.

Step4 (Create TabetaLog): Visualize the JSON data using ActionScript. In the script, the weight records are drawn with a graph, and the pictures are displayed when the user clicks a point on the graph.

4.2 Overview of Experiment

The goal of the experiment is to show the practical feasibility of the proposed LLAPI. In the experiment, five subject implements a program generating the TabetaLog JSON file (via Step 1 to Step 3 of Section 4.1). Specifically, we instruct the subjects to implement two versions of the program: one is with the proposed LLAPI and another is with the conventional API. In the following, let P_{llapi} represent a process (or product) that uses the proposed LLAPI. Also, let P_{conv} stand for a process (or product) that uses the conventional API.

The subjects were instructed to mashup the weight records and the picture records of user "Shimojo" and "Tokunaga" for one year (since 2010-05-18 to 2011-05-17), and to output the resulting JSON file.

Table 2: Result of experiment

Subject	A		B		C		D		E		Correct	
Order of Development	P_{llapi}→P_{conv}		P_{conv}→P_{llapi}		P_{conv}→P_{llapi}		P_{conv}→P_{llapi}		P_{llapi}→P_{conv}		—	
	P_{llapi}	P_{conv}	P_{llapi}	P_{conv}	P_{llapi}	P_{conv}	P_{llapi}	P_{conv}	P_{llapi}	P_{conv}	P_{llapi}	P_{conv}
Programming language	Perl	Perl	Perl	Perl	Java	Java	Java	Java	Java	Java	—	—
SLOC	115	365	227	379	480	612	423	397	150	181	—	—
SLOC(w.out blank and comments)	71	223	103	188	351	426	286	263	106	125	—	—
# of source-code classes	n/a	n/a	n/a	n/a	7	7	5	5	2	2	—	—
# of source-code files	1	4	1	3	n/a	n/a	n/a	n/a	n/a	n/a	—	—
Man-hour (man-minute)	114	196	54	205	96	252	147	514	132	397	—	—
# of weight records `<Shimojo>`	53	54	53	54	32	53	53	54	52	52	53	54
# of weight records `<Tokunaga>`	102	101	102	101	52	103	103	104	103	115	102	101
# of picture records `<Shimojo>`	8	9	8	9	8	9	8	9	8	9	8	9
# of picture records `<Tokunaga>`	85	86	85	86	60	87	85	44	65	85	85	86

Table 3: Evaluation perspectives and metrics

Evaluation perspectives	metrics
Program files	Total lines of codes (SLOC), Number of source-code files (or classes)
Output JSON Files	Number of weight record, Number of picture record
Development Process	Man-hour

4.3 Evaluation Metrics

The experiment has been evaluated from viewpoints of *product* and *process*. Table 3 summarizes the evaluated perspectives and metrics.

The product viewpoint includes two objects: program files (source code) and the JSON file (created data). For the program files, we measure the total lines of codes (SLOC) and number of source-code files (or classes). Also we measure the number of weight records and picture records, to check the *correctness* of source code and JSON file.

As for the development process perspective, we use the metric "man-hour", characterizing effort of the development. After the development, we carried out a questionnaire to the subjects to gather subjective opinions and feedback.

Question 1: Which task did you spend the most time for?

Question 2: How did you feel using the LLAPI?

4.4 Experiment Result

Table 2 shows the results for the five subjects (A, B, C, D, E). For each subject, P_{llapi} and P_{conf} are compared according to the metrics in Table 3. The last column "Correct" represents the correct answers for the generated JSON file, to check the *correctness* of the program.

First, we focus on the total lines of code (SLOC). It can be seen that four of the five subjects decreased the SLOC in P_{llapi}. Specifically, the SLOC of P_{llapi} was 0.72 time smaller than that of P_{conv} on an average.

Next, we compare the man-hour. All of the five subjects decreased the man-hour regardless of the order of the development processes. Especially, subject D achieved P_{llapi} four times faster than P_{conv}. According to the JSON files obtained, we found that only A and B implemented the correct program.

Finally, we show the result of the subsequent questionnaire. Most frequent answer for Question 1 was "Understanding API specification" in both processes P_{llapi} and P_{conv}. For Question 2, the subjects gave positive answers like, "Very convenient" and "Easy to process the multiple lifelog records". We also obtained some comments and requests for the LLAPI including, "Want to obtain contents of the content table by the LLAPI" or "Want to aggregate different usernames for the same user entity".

5. CONCLUSION

To improve the performance and portability, we have re-engineered the lifelog mashup platform [3], consisting of the LLCDM (LifeLog Common Data Model) and the LLAPI (LifeLog Mashup API). We have also evaluated the practical feasibility of the proposed method through development experiment. It was shown in the experiment that the proposed method can efficiently reduce the development effort of the mashup application. Also further efficiency is expected when a more number of services are integrated.

Our future work includes investigation of ontology-based approach and enhancement of the LLAPI for more sophisticated data integration. We also plan to study potentials of lifelog mashups for business and education.

6. ACKNOWLEDGMENTS

This research was partially supported by the Japan Ministry of Education, Science, Sports, and Culture [Grant-in-Aid for Scientific Research (B) (No. 23300009), Young Scientists (B) (No. 21700077), Research Activity Start-up (No. 22800042)], and Hyogo Science and Technology Association.

7. REFERENCES

[1] J. Gemmell, G. Bell, and R. Lueder. MyLifeBits: A personal database for everything. *Comm. ACM*, 49:88–95, January 2006.

[2] J. Gemmell, G. Bell, R. Lueder, S. Drucker, and C. Wong. MyLifeBits: Fulfilling the memex vision. In *Proc. ACM Int'l Conf. Multimedia*, pages 235–238, 2002.

[3] A. Shimojo, S. Kamada, S. Matsumoto, and M. Nakamura. On integrating heterogeneous lifelog services. In *Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services*, iiWAS '10, pages 263–272. ACM, 2010.

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