THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

TECHNICAL REPORT OF IEICE.

Evaluating Lifelog Common Data Model and Mashup API through Practical Application Development

Akira SHIMOJO†, Shinsuke MATSUMOTO†, and Masahide NAKAMURA†

† Kobe University Rokkoudai 1–1, Nada, Kobe, Hyogo, 657–8531 Japan
E-mail: †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 the lifelog common data model (LLCDM) and the lifelog mashup API (LLAPI). In this paper, we evaluate the efficiency of the LLCDM and the LLAPI through a practical application development project. Specifically, we implement two versions of an integrated lifelog application with and without the LLCDM and the LLAPI. We compare the two versions from viewpoints of the quality of the product as well as the efficiency of the development process.
Key words Lifelog, Mash up, Common data model, Web service, Database

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]. 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* [2] for delivering one's status known as tweets, *Flickr* [3] for storing and sharing pictures, *FoodLog* [4] for photographing daily meals, *GARMIN Connect* [5] for managing training data with GPS.

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 record. 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, We presented the *lifelog common data model (LLCDM)* and the *lifelog mashup API (LLAPI)* in the previous work [6] [7]. 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.

In this paper, we evaluate the efficiency of the LLCDM and the LLAPI through a practical application development project. Concretely, we implement a mashup application called *Tabetalog* integrating two heterogeneous services: service that records picture called *Flickr* and the service records weight and body fat percentage called *BodyLogService*. The application is developed in two ways: one process with the conventional APIs, and another process with the LLCDM and LLAPI.

2. PRELIMINARIES

2. 1 Lifelog

As introduced in Section 1., there are a variety of lifelog services available in the Internet. The great success of these services lies not only in information technologies, but also in the nature of human beings, loving to collect and store possessions, memories, experiences [1].

For these lifelog services, "what to log" is a 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.

2. 2 Lifelog Mashup

Integrating such scattered lifelogs may create more values

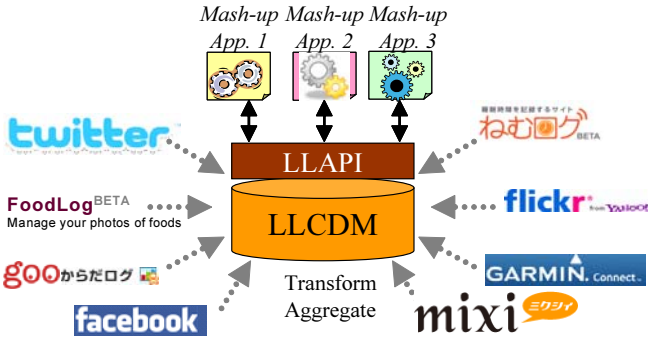

Figure 1 Mashup Approach Proposed in [6] and [7]

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* [8], a doctor may find a symptom of lifestyle related disease by analyzing statistics on eating habits and body measurements.

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.

3. PREVIOUS WORK

To support efficient lifelog mashup, we presented the *lifelog common data model (LLCDM)* and the *lifelog mashup API (LLAPI)* in the previous work [6] [7]. Figure 1 shows the overview of the LLCDM and the LLAPI.

3.1 LifeLog Common Data Model(LLCDM)

The data stored in heterogeneous lifelog services are transformed and aggregated in the LLCDM, which is an application-neutral form among the lifelog services.

Table 1 shows the data schema of the LLCDM. We arranged the data items which lifelog records should prepare from the viewpoints of what, why, who, when, where and how. Then, we defined the "common data store" which does not depend on specific service and application.

We keep the lifelog data conformed to the LLCDM by a relational database. This maintains the accuracy and the access speed of reference, and the performance and the reliability.

3.2 LifeLog Mashup API(LLAPI)

LifeLog Mashup API(*LLAPI*) is for searching and retrieving lifelog data conforming to the LLCDM, which can be used extensively for building lifelog mashups.

```
getLifeLog(s_date, e_date,  s_time, e_time,  user,
                location, application, device)
Paramaters:
  s_date:          Query of <date>
  e_date:          Query of <date>
  s_time:          Query of <time>
  e_time:          Query of <time>
  user:            Query of <user>
  party:           Query of <party>
  object:          Query of <object>
  location:        Query of <location>
  application:     Query of <application>
  device:          Query of <device>
```

Table 1 Common Data Schema of LLCDM

perspective	common data items	explain
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	<address>	Street address where the log is created
	<latitude>	Latitude where the log is created
	<longitude>	Longitude 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

Developer uses the LLAPI for implementing mashup services or applications. Using the LLAPI, all of lifelogs can accessed by common APIs without any knowledge of each lifelog record format.

Currently, we implemented the LLAPI as a Web service (SOAP/REST) so that a lot of programming languages and platforms are able to access the LLAPI.

4. BUILDING A MASHUP APPLICATION

In this section, we conduct an experimental evaluation of developing a practical lifelog mashup application, called *TabetaLog*, by integrating two heterogeneous services: Flickr and BodyLogService. To evaluate the effectiveness of the LLCDM and LLAPI, TabetaLog is developed in two ways: the one process with the conventional APIs, and the another process with the proposed LLCDM and LLAPI.

4.1 TabetaLog

TabetaLog is an application supporting eating habits, which visualizes weight graph and pictures recorded in everyday. This application obtains the two kinds of record from Flickr and BodyLogService respectively, and integrates the record according to date.

Users take daily food photographs by mobile devices (e.g.,

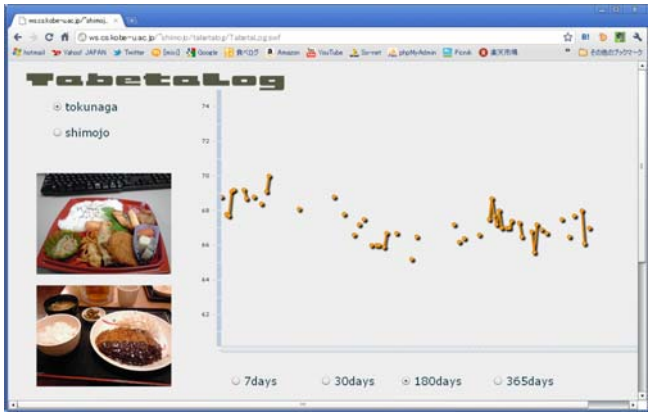

Figure 2 TabetaLog

digital camera, PDA, mobile phone, etc.,). The photographs are uploaded to Flicker. Also, users weigh him/herself once a day using a body scale and register the records to BodyLogService. Next, TabetaLog obtains these records from Flickr and BodyLogService, integrates the records, and outputs the result to a file in JSON format. Finally, using ActionScript, the JSON data is visualized as integrated graph which has a weights graph and food photos.

A screenshot of TabetaLog is shown in Figure 2. Figure 2 shows the weight graph of a user "Tokunaga" for 180 days. When a user clicks a day on the graph, the picture which took the pointed day displayed in lower left of the screen. A granularity of graph points is selected by bottom radio buttons (7days, 30days, 180days or 1year). Also users can see other user's records.

4. 2 Mashup Services

We explain two web services, Flickr and BodyLogService, mashuped by TabetaLog.

4. 2. 1 Flickr

Flickr is an online picture album service provided by Yahoo! Inc. We can arrange, classify and/or exhibit photographs taken by ourself and see other users' photographs, and comment on the photos on the web.

At present, about 180 kinds of API are being shown in Flickr, and it becomes possible to search and obtain the picture and the user, etc. from various conditions[1].

4. 2. 2 BodyLogService

BodyLogService is a weight control service developed in our laboratory. We are measuring daily weights and body fat percentages using BodyLogService.

Currently, five kinds of BodyLogService API are available for obtain the measured weight record. These API supports weight record obtaining by user and by date.

[1]http://www.flickr.com/services/api/

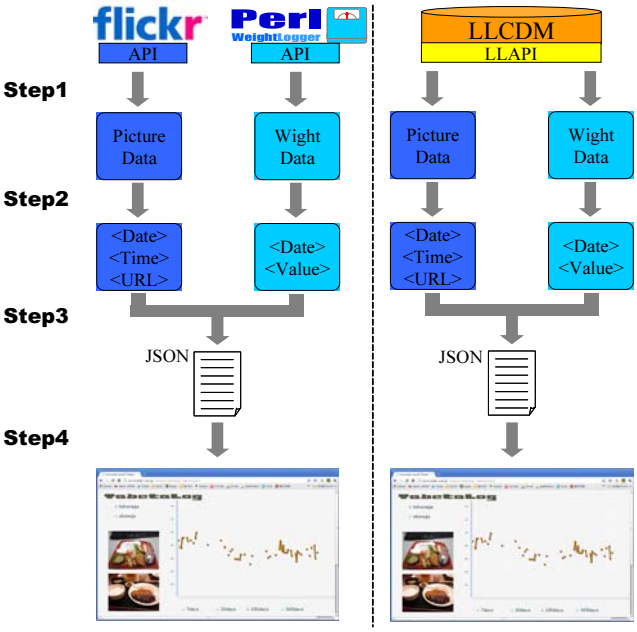

Figure 3 The Process of Implementing TabetaLog

4. 3 Implementation of TabetaLog

We explain the process of implementing TabetaLog. Figure 3 shows the process of implementing TabetaLog. The process consists of the following four steps:

Step1 (Obtain original lifelog record) : Obtain lifelog record using APIs or Web services provided by Flickr and BodyLogService or 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. Note that a joining key is based on date values of weight record. Therefore, if Flickr's record existed but BodyLogService's record didn't exist on one day, we ignore the Flickr's record. Finally, the joined records are dumped to a file in JSON format.

Step4 (Create TabetaLog) : Visualize the JSON data using ActionScript. A weight record is displayed in a graph, and a picture record is displayed when a user clicks the same date.

5. EVALUATION EXPERIMENT

In this section, we explain an experiment of evaluating the effectiveness of LLCDM and LLAPI. In the experiment, all subjects create an input data file for TabetaLog with LLAPI and without LLAPI. Then, we evaluate the efficiency of development from qualitative perspective and quantitative perspective.

```
{
    number: 56,
    user: "shimojo",
    start: "2010-05-18",
    end: "2011-05-17",
    data: [
          {
              date: "2010-09-07",
              weight: "63.4“,
              foods: [
                    {
                        URL: "http://farm5.static.flickr.com/
                              4113/4966191951_7edf219990_m.jpg",
                        time: "13:17:28",
                    },
                    {
                        URL: ···,
                        time: ···,
                    }
              ]
          },
          {
              ···
          },
          ···
    ]
}
```

Figure 4 JSON Format

```
* Quantitative
    Program files
       - Total lines of codes (SLOC)
       - Number of source-code files (or classes)
    Development process
       - Man-hour
    JSON Files
       - Number of weight record
       - Number of picture record
* Qualitative
    Questionnaire
       - Usability of LLAPI
       - Review of development
```

Figure 5 Evaluation Perspectives and Metrics

Table 3 A kind of Flickr API

subject	Flickr API
A	photos.search
B	people.getPublicPhotos
C	photos.search / photos.getInfo
D	photos.search / photos.getInfo
E	photos.search

5.1 Overview of Experiment

In the experiment, three master students and two teachers participate. All of five subjects belong to department of information science, and they are well-trained programmers.

They obtain the records which are necessary for TabetaLog implementation with following two ways and output that in a JSON format (from Step1 of Section 4 to Step3).

P_{LLAPI} : Using the LLAPI

P_{CONV} : Using the proprietary APIs

They obtain weight record and picture record of a user “Shimojo” and “Tokunaga” for one year (since 2010-05-18 to 2011-05-17) in JSON format explained at Figure 4. They develop a mashup program by two ways P_{LLAPI} and P_{CONV} for comparing and conventional method. The order of development way is predetermined for avoiding a habituation of development. Two people are P_{CONV} first, and P_{LLAPI} second, and three people are P_{LLAPI} first, and P_{CONV} second.

They freely develop by their favorite programming language, APIs and experiment environment. However, they have to record the time of experiment using *TaskPit* [2] which is a tool for measuring the PC tasks automatically.

Finally, we carry out a questionnaire to subjects. The question includes about usability of LLAPI and impression of experiment.

5.2 Evaluation of Development

To evaluate the development process and product, we decide to evaluate from the viewpoint of quantitative and qualitative. “Product” includes two object, the one is the JSON file that is output result and another is source-code files which create the JSON file. Evaluation perspectives and metrics are shown in Figure 5.

[2]http://taskpit.jpn.org/index.html

From the quantitative viewpoint, we measure total lines of codes (SLOC) and number of source-code files (or classes). These metrics are generally used as a criterion to describe a scale of development product. We also measure man-hour that includes all time spent for the development (programming, testing, searching API specification, etc.,). Also we measure number of weight records and picture records to check the accuracy and correctness of source code and JSON file. This metrics represents that generated JSON file does not have duplications or missing of records.

From the qualitative viewpoint, we carry out a questionnaire in the last of the experiment, and demand an opinion about the impression through the experiment or the thing that they noticed.

5.3 Resullt

We show the result of experiment in Table 2. “Correct” row in Table 2 means corrected answers for generated JSON file. The reason why the corrected answers are different in P_{LLSPI} and P_{CONV} is that there were some duplication and some missing in database of LLCDM. “Order” line in Table 2 means which process are conducted at first. In addition, we show a kind of Flickr API used by each subject in Table 3. Results of each questionnaire are shown in Table 4 to Table 6.

From a quantitative viewpoint, four of five subjects decreased total lines of codes of the source-code files in P_{LLAPI}.

Table 2 Result of Experiment from Quantitative Viewpoints

Subject	Correct		A		B		C		D		E	
Order	—		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}
Program Development												
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
Mashup JSON Data Obtained												
# of weight records `<Shimojo>`	53	54	53	54	53	54	32	53	53	54	52	52
# of weight records `<Tokunaga>`	102	101	102	101	102	101	52	103	103	104	103	115
# of picture records `<Shimojo>`	8	9	8	9	8	9	8	9	8	9	8	9
# of picture records `<Tokunaga>`	85	86	85	86	85	86	60	87	85	44	65	85

Table 4 Question1. Which tasks did you spend the most time? (multiple answers allowed)

	P_{LLAPI}	P_{CONV}
Understanding API specification	2	3
Obtaining lifelog record	0	1
Extracting data records	0	1
Joining data records to JSON	1	0
Others	1	0

This means product scale has become about from 0.3 times to 0.8 times compared with in P_{CONV}. Moreover in P_{LLAPI} subject A and B who are Perl programmers completed the experiment from a single source-code file. However, they need to create four or three sources in P_{CONV}. This difference was not seen in Java developers.

All of five subjects decreased the man-hour whichever they implement first by P_{LLAPI} or P_{CONV}. Especially, subject D decreased one-quater compared with P_{CONV}.

About mashup JSON files, only two subjects obtained the correct record that both of the user "Shimojo" and "Tokunaga" whichever by P_{LLAPI} and P_{CONV}.

From a qualitative viewpoint, we carried out questionnaires. Answers of a question "Which tasks did you spend the most time?" is shown in Table 4. Most frequent answer was "Understanding API specification" both of P_{LLAPI} and P_{CONV}. In P_{LLAPI} a subject answered "Joining data records to JSON", however in P_{CONV}, subjects answered "Obtaining lifelog record" and "Extracting record items".

Secondly, we asked "Impression through an experiment" (shown in Table 5). They answered "Very convenient", "the development effort and the amount of coding becomes small" and "easy to process the multiple lifelog records". Also some requests for LLAPI obtained. These are "Want to obtain contents of the Content table by LLAPI" or "Want to absorb the difference of user-names for each lifelog service".

Finally, the question "Impression through an experiment (Table 6)", there were a lot of impression that it took time to a part different from the essence of the experiment such as "I had a hard time to generate JSON file".

5. 4 Discussion

We discuss about the result of an experiment in 5. 3.

5. 4. 1 Development Efficiency of Application

Compared with conventional method, the efficiency of mashup development using LLAPI is clearly improves from the quantitative results in Table 2, and the questionnaires in Table 5 and Table 6. In the experiment, we integrated two heterogeneous lifelog services. However we think that the efficiency would be more remarkable if a greater variety of lifelog services required. Because understanding API specifications require much effort.

5. 4. 2 The Accuracy of the Data Obtained

From accuracy perspective, the correctness of generated JSON file are same in P_{LLAPI} and P_{CONV}. We expected that implementing with P_{LLAPI} improves the correctness compared with P_{CONV} because source codes become simple and small. We guess this cause to be that LLAPI didn't support logics such as joining the record by date. In this case, the subjects were necessary to obtain the two kinds of lifelog records Flickr and BodyLogService. However, LLAPI currently does not have a method that joins multiple record by date, user or application. Therefore, subjects need obtain two lifelog records separately and need to create a logic of record joining in both cases.

5. 4. 3 Limitations

At first, there is a limitation about the flexibility of LLAPI. As explained in 5. 4. 2, LLAPI must support mashup request such as obtaining a variety of lifelog records at single API call to adapt a wide variety of lifelog services. For that reason, there is our future work that implementing more LLAPI functions for obtain multiple lifelog records at the same time or obtain by same date,

Secondly, we consider about preserving method of the con-

Table 5 Question2. How did you feel using LLAPI?

- Second task P_{CONV} could easily develop by reusing source codes written at first task P_{LLAPI}. However, I spent great deal of effort to understand Flickr API specifications.
- I thought that the difference of development effort becomes extremely large if three or more lifelog services mashuped.
- I felt LLAPI was very convenient. LLAPI was not necessary to use many APIs like Flickr because all data displayed by the list I wanted to obtain.
- I thought it was better when contents of the Content table were possible to get by LLAPI.
- LLAPI was very convenient, but I doubted it reliability and real time.
- Because the API and the data format retrieved were standardized, it was able to process the multiple lifelog records easily.
- I wanted the LLAPI to absorb the differences of the user name at each service.
- I want to obtain the mashuped data by specifying some user names or some application names. For example, getLifeLog(..., application=Flickr,BodyLogService) is greate.

Table 6 Question3. Impression through the experiment

- I had trouble with a few part in the essence of experiments such as the JSON conversion.
- Because the preliminary knowledge was personally insufficient, it took very time.
- I had a hard time because I had made the generation of the JSON file by itself.
- I was bewildered to the results retrieved each way was different.
- It was difficult to find the API because Flickr provides too many APIs.

tent table. Currently, whole of the content table are same as original record without processing or revising. So LLAPI user has to understand the structure of original record when they want to obtain the data (e.g., picture URL of the Flickr). However, because the structures of original record are completely different for each lifelog service, it is very difficult to define data schemas. Therefore, we are considering the storage methods that use a schemaless approach such as NoSQL and KVS.

Finally, we have to develop a method that absorbs the difference of the username varied for different services. There isn't uncommon that a user registers a different usernames for difference services. For example a user "Shimojo" has a different username, "jo_shimo23" as a Flickr name and "Shimojo" as a BodyLifeLogger. If lifelog service increases, a variety of user names are increased. To solve such a problem, LLCDM should have user definition table and supports API calling by using representative user names such as Single Sign-On. For example, if a user only specifies "Shimojo", LLAPI respectively interprets "Shimojo" and "jo_shimo23" for Flickr and BodyLifeLogger.

6. CONCLUSION

In this paper, we have evaluated the efficiency of the LLCDM and LLAPI through a practical application development project. We experimented the developing mashup application TabetaLog by integrated two heterogeneous services, Flickr and BodyLogService to five subjects. In this experiment, we confirmed the efficiency of the LLCDM and LLAPI and it seems that the effectiveness of LLCDM and LLAPI rises further while the life log service will increase in the future.

Our future work is to relax the limitation presented in the previous section. After the limitations are resolved, we investigate the additional values of lifelog mashups.

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.

References

[1] Trend Watching .com, "Life caching – an emerging consumer trend and related new business ideas". http://trendwatching.com/trends/LIFE_CACHING.htm.
[2] Twitter, Inc., "twitter". http://twitter.com/.
[3] Yahoo, "Flickr". http://www.flickr.com/.
[4] K. Kitamura, T. Yamasaki, and K. Aizawa, "Foodlog: Capture, analysis and retrieval of personal food images via web," CEA '09: Proceedings of the ACM multimedia 2009 workshop on Multimedia for cooking and eating activities, pp.23–30, ACM, New York, NY, USA, 2009.
[5] Garmin Ltd., "Garmin". http://connect.garmin.com/.
[6] A. Shimojo, S. Kamada, S. Matsumoto, and M. Nakamura, "On integrating heterogeneous lifelog services," Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services, pp.263–272, iiWAS '10, ACM, 2010. http://doi.acm.org/10.1145/1967486.1967529
[7] S. Akira, M. Shinsuke, and N. Masahide, "Implementing database and web services for life-log mashup apis [in japanese]," IEICE technical report, vol.110, no.282, pp.101–106, 2010-11-10. http://ci.nii.ac.jp/naid/110008153938/en/
[8] NTT Resonant Inc., "karadalog". http://karada.goo.ne.jp/.