Abstract—In this paper, we aim to overcome barriers to realizing practical lifelog systems, and we propose a lifelog system named ‘LifeLogOn’ (Lifelog Ontology; ‘Log on to your lifelogs’). The system allows users to easily build and exploit lifelog ontology by importing various available data sources. In addition, we present some use case scenarios to show how our system and the lifelog ontology built by our system can be used for many purposes.

I. INTRODUCTION

A Lifelog system can be defined as the system which involves gathering and exploiting any historical/personal activity records, which are called lifelogs. A lifelog system which can support building and using high quality and large size of lifelogs can bring us many advantages in our lives. For example, when someone does not remember important things that the person has interacted with, he/she can ask the lifelog system, and the system can answer what he or she wants to find out referring to the lifelogs exploited by the system. Additionally, lifelogs gathered by a lifelog system can be utilized as source for data mining. A wide range of data mining techniques can be applied to lifelogs to find out useful information for many applications (e.g. a person’s characteristics, preference, and so on.)

Many of earlier lifelog systems are influenced by the vision of the Memex which is proposed by Vanevar Bush in 1945. He introduced a conceptual device called Memex, where an individual stores all of his personal information [12].

The ideal goal of life logging is to record everything about a person. Most of the current existing systems had difficulties and remain as research projects, because they tried to achieve the goal of life logging by adapting technologies that have high complexity (e.g. physical sensors, video recognition, high-level artificial intelligence) [9], [5], [10], [16], [14] or large size of data storage [3]. Although flexibility is very important factor for a lifelog system, many of systems have neglected flexibility problems. Some systems depend on specific application, device, or data schema [11], [1], [15], or some other systems require additional code-level efforts for processing new data sources [12], [7].

Today, people continuously produce a lot of information that can be used to describe their lives. For instance, a person produces SMS message, email and phone call logs every day. This gives us hints let us conclude that integrating the available logs from various devices and creating semantic relationships among can be a starting point of realizing a practical lifelog system rather than relying on highly complex technologies.

Let us say you are trying to find out the song you listened to at a party with your friends years ago, when only things you have are the photos that you took at the party. To find the song, a lifelog system may try to recognize objects in the photos, find out who and what are in the photos, represent the relationships of the items and music preferences in the photo with ontology languages, and perform inference with facts available to them. It does not seem to be a practical solution, because it requires high computational cost. Unfortunately, many earlier systems made the problem more complex, in that they approach like the scenario above. However, the song could be easily found by linking the photos and music listing logs which have the same date information. It’s much simpler and practical at the same time. Many other needs in real-life can be fulfilled by this simple solution.

Based on the idea, we propose a prototype of LifeLogOn system (Lifelog Ontology; ‘Log on to your lifelogs’) that enables users to easily import various source data into integrated lifelog ontology and navigate both lifelog schema and instances in the lifelog ontology by following their semantic relationships. To achieve flexibility of LifeLogOn, we adapted the Entity-Event Lifelog Ontology Model (EELOM) [7] for modelling the LifeLogOn’s lifelog ontology. EELOM’s flexibility allows users to easily extend the lifelog ontology without writing additional codes when they need to process new types of data sources. LifeLogOn’s approach could make up for the difficulties of existing approaches. We have demonstrated earlier version of our system [6].

In section 2, we briefly introduce related works. Section 3 explains the overview of LifeLogOn including EELOM, our assumptions, and LifeLogOn’s architecture. In section 4, 5 we explain the overall processes of building and exploiting lifelog ontology with LifeLogOn by giving some example scenarios. In section 5, we conclude our work and present future works.

II. RELATED WORK

As we already mentioned in the introduction, there have been various forms of research projects aiming to record personal information. Nokia¹’s Lifeblog, Nike+² are examples of the approach that is to support specific devices or applications.

¹Nokia Lifeblog, http://webphones.nokia/blog/
²Nike+, http://nikerunning.nike.com/nikiplus/
Researchers also have proposed more general systems that are not dependent on specific devices or applications, and which can handle lifelogs covering various data types. MyLifeBits [3] aims to record every information of a person including contacts, calendar, documents, and so on. The vision of MyLifeBits which is to store ‘everything’ in a solipsistic system is the ideal goal of lifelog system; thus, the system remains as an experimental project than a practical system. Memmel et al. [11] presented another earlier framework to support storing and retrieving personal information of users; however, it only supports fixed data types, so it is not possible to process new data type is available.

As semantic technology is introduced in the world, some projects (e.g. SemanticLife [1] and Semantic Logger [15]) adapted ontology languages such as RDF³, OWL⁴ for modeling lifelogs due to their expressiveness. SemanticLife and Semantic Logger require additional code writing when a new data source needs to be emerged in the system. Thus adding support for new data sources to them is time consuming task.

Several lifelog systems [9], [5], [10], [16], [14] utilize various hardware and software technology such as wearable computers and smart objects. Kim et al. [5] uses video recognition technology to automatically annotate human activities in the video; however, the assumption of equipping wearable computers and high complexity of adapted technologies make the systems remain as research practices rather than practical systems.

III. LIFELOGON: OVERVIEW

Through the survey of earlier lifelog systems, we could derive two critical considerations for developing practical lifelog systems. The first one is flexibility, which means that a practical system should be able to process a wide range of data sources, and adding additional data source should be easy. Nowadays, new devices and data sources are introduced every day. if adding new data source is difficult or not possible for a lifelog system, the lifelog cannot be considered as a practical system.

The second important requirement is that a practical lifelog system should not depend on complex technologies which require high computational cost. Recall the music-finding example presented in the former section. Even if facts that indicate photo-taking and music-listening are connected by lengthy chain of references, performing inference until the system finds the information requires long computation time and clever rules defined by thoughtful experts which naive users would not be able to provide by themselves.

By considering these two, we developed a practical lifelog system. In this section, we present the overview of our system named LifeLogOn (Lifelog Ontology; ‘Log on to your lifelogs’). LifeLogOn shows that we could build a large size of useful lifelog ontology instances only by integrating currently available data produced by heterogeneous devices and applications.

³Resource Description Framework, http://www.w3.org/RDF/
⁴Web Ontology Language, http://www.w3.org/2004/OWL

In the following, we present the Entity-Event Lifelog Ontology Model (EELOM) that is used as lifelog ontology model in LifeLogOn, our assumptions, and the system architecture of LifeLogOn.

A. Entity-Event Lifelog Ontology Model (EELOM)

Figure 1 shows the bird-eye overview of EELOM. As depicted in the figure, EELOM consists of domain, entity events, and their relationships. The advantages of EELOM can be summarized into two – simplicity and flexibility. Firstly, EELOM is very simple data model, thus it is easy to learn. At the same time, it is expressive enough to cover various data sources. Secondly, EELOM supports flexibility which is one of the most important requirements of a practical lifelog system. EELOM is not dependent on specific applications or devices and can be easily extended by adding new events or entities.

B. Assumptions

We developed LifeLogOn based on the several assumptions that are carefully made to make the system more practical and general at the same time.

Firstly, we assume that all of the source data can be represented in relational model. So, if users want to import data that is in other formats (e.g. XML, spread sheets), then they need to transform the source data and store those into relational databases. We insist this assumption is reasonable, because relational database is the most common data storage, and most of source data can be transformed into relational data without too heavy effort. Any kind of data can be imported to LifeLogOn, if the data is considered to be good to describe the user’s life and if they are stored in relational databases.

The second assumption is that LifeLogOn imports only two categories of data – Log type and Meta type – for building or enriching its lifelog ontology. Log type data is records of historical events of any activities. This type of data can be used to enrich lifelog events in lifelog ontology. Played song history of Last.fm⁵, location tracking logs gathered by a GPS device, phone call history from a smart phone can be examples of log type data. Meta type data is data that

⁵Last.fm, http://www.last.fm/
describe entities in the world. This type of data can describe the details of entities in the lifelog ontology. For example, music metadata database such as All Music Guide (AMG)\(^6\) or CDDB\(^7\) could be imported to describe song entities in the ontology. LifeLogOn does not store multimedia contents in its knowledge base.

Lastly, we assume that the user has ability to design the lifelog ontology schema for the user’s purposes of use. LifeLogOn automatically generates lifelog ontology instances; however, the generation is based on the lifelog ontology schema and source data to the schema mapping information given by the user. To use LifeLogOn, users should understand the schema structure of source data and EELOM model, so that they can define lifelog ontology schema to cover all the information that the user wants to import. This assumption may invite a criticism that an ordinary person who is not familiar with database systems generally does not understand the schema of source data and EELOM. This is correct, but LifeLogOn targets the users who want to get ‘Lifelog data (ontology)’ not ‘Lifelog services.’ We assume that people who want to get ‘Lifelog data’ have more knowledge of source data. Also, we assume that EELOM is simple and easy to learn enough for such people.

C. System Architecture

LifeLogOn consists of two main parts – LifeLogOn Tool and LifeLogOn Knowledge Base. LifeLogOn Tool is the part of the system that end-users operate. It contains five submodules – ‘Lifelog Ontology Schema Definition Module,’ ‘Source Data to Lifelog Mapping Module,’ ‘Lifelog Ontology Instantiation Module,’ ‘Search & Visualization Module,’ ‘RDF Export Module,’ and a unified web-based graphic user interface. LifeLogOn Knowledge Base is the storage where lifelog ontology schema and instances are stored. It also contains mapping information between source data and lifelog schema. Lifelog Indexer Module maintains the index structures for fast access to ontology instances, and Inference Module supports simple inferences. Figure 2 depicts the LifeLogOn’s architectural design.

IV. BUILDING LIFELOG ONTOLOGY USING LIFELOGON

Figure 3 shows the whole process of building lifelog ontology using LifeLogOn. Each step is performed by collaboration of several sub-modules in LifeLogOn’s architecture. The LifeLogOn provides intuitive web-based graphic user interface to help users to easily handle each. On the background, the knowledge base continuously manages lifelog ontology instances, and index structures for fast retrieval. The user can import and accumulate new types of data sources by restarting the process. In the following paragraphs, we explain the details of each process using an example use case.

\(^7\)http://www.gracenote.com
When storing the schema, LifeLogOn automatically gives identifier values to each attribute, entity, and event. LifeLogOn also allows users to visualize the ontology schema when they are defining it. If users click the visualization button on the bottom, the system graphically shows the ontology schema by running the visualization module.

We give an example of lifelog ontology schema that covers example source data depicted in Figure 4. We define a lifelog ontology schema consisting of five domains – Music, DigitalCamera, Email, Phone, and Calendar. Each domain consists of several entities and events. For instance, DigitalCamera domain is composed of two events (TakePhoto, BrowsePhoto) and five entities (User, Time, Date, Location, PhotoEntity) represent the TakePhoto event and BrowsePhoto event. By sharing the same entities, events in different domains have relationships. For example, as depicted in Figure 5, the TakePhoto has a relationship with the MusicPlay in that they share the same entities (User, Location, Time, Date). Similarly, entities have relationships when they have connections to each other. For example, PhotoEntity and MusicEntity have relationship in that they are indirectly connected to each other following links (PhotoEntity - TakePhoto - Time - MusicPlay - MusicEntity). It is important that lifelog ontology instances inherit the relationships.

3) Mapping Source Data to Lifelog Ontology Schema:
As the next step, the user lets the LifeLogOn know which attribute of source data table has the same semantics as an attribute in the defined lifelog ontology schema, so the user maps source data schema to lifelog ontology schema. ‘Source Data to Lifelog Mapping Module’ supports management of mapping information profiles, so that users can add, edit, and delete profiles.

As we explained, LifeLogOn can import two types of data – Meta type & Log type. Because Meta type data means the data that describes entities in the ontology schema. We map Meta type source data table’s attributes to attributes of an entity in the ontology schema. Mapping profile of Meta type source data tables to ontology schema O can be defined as follow:

\[MP_{\text{meta}}(O,T_{\text{meta}}) = \{(a_1,t_1,s_{a1}),\ldots,(a_i,t_i,s_{ai}),\ldots,(a_n,t_n,s_{an})\},\]

where \(a_i\) represents an attribute of an entity in O, and \(s_{ai}\) represents one attribute of \(t_i\) and \(t_i \in T_{\text{meta}}\).

On the other hand, mapping profiles for Log type data should contain more information. Log type data is the data that contains a set of historical events, and we assume that a tuple of the data can be mapped to an event in the ontology schema. So, we map Log type data table’s attributes to attributes of entities that describe an event. When one log table contains one or more than one kinds of events, we need to let LifeLogOn know what kind of event the each tuple represents. In this case, a pair of attribute and its value distinguishes an event type from the other event types. We call these identifier attribute and value. For example, we can distinguish what kind of event type a tuple from LOGDATA_MUSIC table represents by seeing whether the value of type attribute is play or stop. Thus, attribute-value pair of type-play or type-stop is considered as a pair of identifier attribute and value. Mapping profile of Log type source data tables to ontology schema O can be defined as follow:

\[MP_{\text{log}}(O,T_{\text{log}}) = \{(a_1,t_1,s_{a1},i_{a1},i_{v1}),\ldots,(a_i,t_i,s_{ai},i_{ai},i_{vi}),\ldots,(a_n,t_n,s_{an},i_{an},i_{vn})\},\]

where \(a_i\) represents and attribute of an entity that describes an event in O and \(s_{ai}\) represents one attribute of \(t_i\) and \(i_{ai}\) and \(i_{vi}\) represents identifier attribute and value of \(t_i\) and \(t_i \in T_{\text{log}}\).

Figure 6 shows an example of mapping between an lifelog ontology schema and source table METADATA_MUSIC and LOGDATA_MUSIC.

4) Generate Lifelog Ontology Instances: Based on the given source data tables, and a lifelog ontology schema, and their mapping profiles, LifeLogOn’s Lifelog Ontology Instantiation Module automatically generates lifelog ontology instances and stores the instances into the LKB. These instances are based on EELOM, but LifeLogOn also stores these

![Image 4. Example source data of lifelog ontology building](image)

![Image 5. Part of defined lifelog ontology schema](image)
instances in general triple data model, too. By storing instances based on two different models, LifeLogOn can take advantage of EELOM and general triple data model at the same time. Figure 7 shows the overall process of lifelog instantiation in LifeLogOn. At first, LifeLogOn’s Instantiation Module instantiates entity instances using the given ontology schema $O$, a set of Meta type source tables $T_{meta}$, and mapping profiles between them $MP_{meta}(O, T_{meta})$. The instantiation algorithm is presented below:

```plaintext
for all $table_i \in T_{meta}$ do
    for all $tuple_j \in table_i$ do
        $R \leftarrow$ FindMappingRules($MP_{meta}, O, tuple_j$)
        $e \leftarrow$ MakeEntity($R, tuple_j$)
        $LKB \leftarrow LKB \cup e$
    end for
end for
```

As the next step, LifeLogOn’s Instantiation Module instantiates event instances using the given ontology schema $O$, a set of Log type source tables $T_{log}$, and mapping profiles between them $MP_{log}(O, T_{log})$. Following shows the instantiation algorithm:

```plaintext
for all $table_i \in T_{log}$ do
    for all $tuple_j \in table_i$ do
        $E \leftarrow$ ExtractEntities($O, tuple_j, MP_{meta}$)
        for all $e \in E$ do
            if $e \not\in LKB$ then
                $LKB \leftarrow LKB \cup e$
            end if
        end for
        $R \leftarrow$ FindMappingRules($MP_{log}, O, tuple_j$)
        $l \leftarrow$ MakeLogInstance($R, tuple_j$)
        $LKB \leftarrow LKB \cup l$
    end for
end for
```

Event instantiation process is similar to the entity instantiation process; however, it requires additional jobs to create entity instances that are required to instantiate events that tuples represent. LifeLogOn checks if there already exists the same entity instances in LKB, so that it can prevent duplicate instances in LKB. However, deciding whether two entity instances are the same is not straightforward, when they are very similar but little different. Although the current prototype of LifeLogOn cannot deal with it, we are planning to tackle this problem by adapting similarity measures in the next version of LifeLogOn.

LifeLogOn uses the created EELOM instances as an input to create triples. Finally, LKB’s Lifelog Indexer Module constructs index structures for the generated instances. The overall ontology instantiation process is summarized in the Figure 7.

LifeLogOn uses the created EELOM instances as an input to create triples. By analysing the EELOM instances, the system creates nodes (entity, attribute, event, and value), and triples that are composed of them. Finally, LKB’s Lifelog Indexer Module constructs index structures for the generated instances.

V. EXPLOITING LIFEOLOGY ONTOLOGY USING LIFEOLOGY ON

The lifelog ontology instances generated by LifeLogOn are semantically enabled in that we can exploit the relationships among events and related entities in the ontology. However, generally, a lifelog ontology instances are a lot of entities, events, and their relationships, so we need to some efficient ways of accessing instances. Thus, we developed a Search and Visualization Module using RaVis library\(^8\) for enhancing the accessibility of users to instances that they want to look at. Finally, users can search and browse their life experiences, or they can exploit the generated ontology instances for their own purposes.

To use the GUI tool, a user controls the inference level by adjusting slide bar and inputs keywords for search. When a user requests a query with keywords and the inference level is set as n, then the system searches and visualizes all the nodes (entities, events, and attributes) that have links in n-hops with the values that match the given keywords. By using

\(^8\)RaVis, http://code.google.com/p/birdeye/wiki/RaVis
A. Use Case I: Finding a song that you don’t remember

In the motivating example presented in the first section, you are trying to find a song you listened to when you took some photos at the party. Viewing photos, you discover one of them has filename of ‘DSC_2128.jpg.’

Thus, you type the filename ‘DSC_2128.jpg’ into the keyword search box of the Search & Visualization Tool. The tool shows events and related entities having values those match the filename. You look at the TakePhoto#467928 event which has a link to the photo entity that has filename of ‘DSC_2128.jpg’, and you realize that the event has a hasDateOf relation to a Date entity whose value is 2009/12/24. You click the entity, and you see many events happened on the day. Among them, you find the MusicPlay#468016 event which has a link to the song entity that has title of ‘Lazy Rhapsody,’ which you were looking for. You can also know that the artist of the song is ‘Duke Elington.’

B. Use Case II: Finding a photo that you took when you were talking on the phone with your friend

During your trip in Korea, you took a photo while you were talking on the phone with your friend, saying ‘It’s so beautiful here!’ Assume that now it has been 2 months since then, and you want to show the photo you took when you were talking on the phone; however, you forgot the exact time and date. Only thing you know is that the friend’s phone number is ‘018-2144-8842.’

So, you type ‘018-2144-8842’ in the keyword search box of Search & Visualization Tool, and the tool shows you a phone call event PhoneCall#465893 and lets you know that you made a phone call to the number in the date of ‘2009/01/04’. You click the date entity ‘2009/01/04’ and the system shows you a TakePhoto#467929 event that has a relation with a photo entity which has the filename of ‘DSC_4204.JPG.’ Finally, you have found the filename of the photo that you were looking for and can show the file to your friends.

The use case scenario I and II show that navigation process following links among events and entities of lifelog ontology instances can effectively support users’ memory, so users can find what they do not remember exactly and even more information, starting from a very small clue. Users can use any information as a clue for searching things, if the clue is semantically related to the things that you are looking for.

C. Use Case III: Supporting Context-Aware Recommendation using Lifelog Ontology

A context-aware personal service actively and autonomously adapts and provides the most appropriate service or content to users, accurately interpreting the context without much user supervision [8]. Performing context-aware recommendation is a good example of context-aware personal service.

Assume that we have a context-aware recommendation engine that uses users’ current context as a query and lifelog ontology instances as target data of the query. When your current location is ‘Times Square, NY’ and current date is ‘2009/12/24’, the engine searches top n entities that you interacted with in the similar context of your current context. For example, the results includes song entities played in ‘12/24’ in another years or movie entities watched near ‘Times Square, NY’. The engine could directly recommend the search results to you, or it could find similar entities to the search results and recommend them to you. The use case scenario presented may be too simple to be used in real life applications; however, it shows that using lifelog ontology can support context-aware personal services. In this paper, we introduce a lifelog system named ‘LifeLogOn’ motivated by the fact that we could support useful applications only by integrating available logs and creating semantic relationships among the log items. Without relying on high complex technologies, LifeLogOn supports building of lifelog ontology and exploiting entities and events in the built ontology. To show how LifeLogOn and the built lifelog ontology can be used, we present several use case scenarios.
LifeLogOn can be enhanced in many ways. We plan to work on adding several features in the next version of LifeLogOn. Firstly, we consider adding data summarization/abstraction feature. Normally, it is very difficult to deal with large numbers of lifelog ontology instances; however, we assume that the data can be summarized or abstracted into manageable numbers of semantic concepts. For example, as seen in the Figure 11, Time entities which have attribute values such as ‘13:00’ or ‘15:00’ can be abstracted into ‘Afternoon’ concept. We believe that not only can we reduce the cost of processing large number of instances but also can we efficiently exploit the implied semantic information in the log instances. Secondly, we plan to develop APIs for enabling other applications to easily access LifeLogOn’s lifelog ontology. If LifeLogOn includes APIs for creating, editing and retrieving lifelog instances, many applications can be easily developed using the APIs and take advantage of LifeLogOn’s lifelog ontology.

ACKNOWLEDGMENT

This research was supported by the MKE(The Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency). (grant number NIPA-2010-C1090-1031-0002)

REFERENCES