UbiqLog: a generic mobile phone-based life-log framework

Rawassizadeh, Reza; Tomitsch, Martin; Wac, Katarzyna; Tjoa, A Min

How to cite

RAWASSIZADEH, Reza et al. UbiqLog: a generic mobile phone-based life-log framework. In: Personal and ubiquitous computing, 2013, vol. 17, n° 4, p. 621–637. doi: 10.1007/s00779-012-0511-8

This publication URL: https://archive-ouverte.unige.ch/unige:72569
Publication DOI: 10.1007/s00779-012-0511-8

Pers Ubiquit Comput (2013) 17:621–637
DOI 10.1007/s00779-012-0511-8

ORIGINAL ARTICLE

UbiqLog: a generic mobile phone-based life-log framework

Reza Rawassizadeh · Martin Tomitsch · Katarzyna Wac · A. Min Tjoa

Received: 2 May 2011 / Accepted: 1 February 2012 / Published online: 3 April 2012

Abstract Smartphones are conquering the mobile phone market; they are not just phones; they also act as media players, gaming consoles, personal calendars, storage, etc. They are portable computers with fewer computing capabilities than personal computers. However, unlike personal computers, users can carry their smartphone with them at all times. The ubiquity of mobile phones and their computing capabilities provide an opportunity of using them as a life-logging device. Life-logs (personal e-memories) are used to record users' daily life events and assist them in memory augmentation. In a more technical sense, life-logs sense and store users' contextual information from their environment through sensors, which are core components of life-logs. Spatio-temporal aggregation of sensor information can be mapped to users' life events. We propose UbiqLog, a lightweight, configurable, and extendable life-log framework, which uses mobile phone as a device for life logging. The proposed framework extends previous research in this field, which investigated mobile phones as life-log tool through continuous sensing. Its openness in terms of sensor configuration allows developers to create flexible, multipurpose life-log tools. In addition to that, this framework contains a data model and an architecture, which can be used as reference model for further life-log development, including its extension to other devices, such as ebook readers, T.V.s, etc.

Keywords Life-log · Mobile sensing · Smartphone · Personal digital memory

R. Rawassizadeh (✉) · A. M. Tjoa
Institute of Software Technology and Interactive Systems Vienna, University of Technology, Favoritenstrasse 9-11/188, 1040 Vienna, Austria
e-mail: rrawassizadeh@acm.org

A. M. Tjoa
e-mail: amin@ifs.tuwien.ac.at

M. Tomitsch
Design Lab-Faculty of Architecture, Design and Planning, The University of Sydney, Sydney, NSW 2006, Australia
e-mail: martin.tomitsch@sydney.edu.au

K. Wac
Institute of Services Science, University of Geneva, Rue de Drize 7, Battelle A, 1227 Geneva, Switzerland
e-mail: katarzyna.wac@unige.ch

1 Introduction

In the past, most life-log tools were custom-built devices or applications using external hardware as sensors (e.g. [1, 2]). With the advent of smartphones, the mobile phone has become a feature-rich mobile device, creating an opportunity to use it as platform for life logging. It is not easy to convince users to carry something other than their usual digital devices such as mobile phones or watches to perform a novel activity that in this case is recording their daily activities. Smartphones are small computers with computing capabilities, but more importantly, they are equipped with sensors and networking technologies, such as Bluetooth, which can sense and communicate with the user's environment. Their networking capabilities enable these devices to be connected to a larger computing or storage media, becoming a hub for personal or body area networks [3]. Further, smartphones can provide useful information about our daily activities such as received and dialed calls, text messages, application usage, phone camera pictures, etc. A disadvantage of dedicated life-log devices is that, users need to carry yet another device with them just for the purpose of recording their daily activities.

Based on smartphone capabilities and features, we believe that they represent a promising platform for life logging. Therefore, the focus of this study was to create a life-log tool that resides in the user's smartphone and does not require any additional hardware (except that hardware is a new external sensor). However, smartphones are not designed to be used as life-log tools, leading to a number of challenges when using them for this purpose. For example, the performance of the phone could be affected by the amount of resources required by a life-log application running on the phone.

Bell et al. [4] introduced some challenges of personal archives, which we call life-log dataset in this article. These challenges include controlling the access to the information and personal security, browsing and searching the archive, and the requirement that the archive should support long-term preservation of information. Log-term preservation is an important requirement, since life-logs are worth to be kept at least as through the owner's life.

Briefly, design requirements of a life-log system are: *seamless integration into daily life*, *resource efficiency*, *security*, *long-term digital preservation*, and *information retrieval* from the life-log dataset.

This research proposes UbiqLog, a life-log framework for smartphones that addresses life-log challenges, while remaining configurable, flexible, and extendable. More on this will described in following sections. It is important to note that the focus is on the collection stage of life logging, and therefore, this research does not delve deep into reflection methods. Although for the purpose of demonstration, we provide visualization features, which will be examined in the evaluation section. Additionally, the prototype features a search facility to enable users manage their life-log dataset and remove undesired information from their dataset directly on their mobile device. This is an important requirement due to the fact that users are the owners of their life-log dataset, and thus, they should have full control on their personal information. Sensors are core components of life-logs. The proposed framework allows users to configure the sensors they want to use for recording their life-logs by disabling or enabling a sensor, or changing a sensor settings. Additionally, users can add a new sensor to the framework or remove an existing sensor from the framework. Flexibility to this extent requires a specific data model and architecture, which we propose as a part of this research. Based on the proposed data model and architecture, we implemented a prototype of a life-log tool running on the Android 2 platform for Android phones. The prototype is flexible enough to be used for other platforms just by changing sensors according to the target platform. So far, Android has been used for mobile phones and tablet computers, but it could be used on other things such as T.Vs[1], and vehicles. Although the implementation have been done on the Android platform, the architecture and data model are not Android-dependent and can be used on other platforms as well.

This work provides a novel approach to life logging by enabling users to configure sensors and add new sensors. Other efforts, which employ mobile phones for life logging, provide a closed architecture, making it difficult or impossible to customize the life-log tool. Related projects that use an open architecture such as MyExperience [5] are not designed for life logging. Their focus is on other research aspects such as context sensing, and to our knowledge, there is no other open architecture for a life-log framework available.

Since life-log systems are similar to context-aware systems, it is notable that a life-log tool is different from a context-aware tool in several aspects. For example, life-log tools should archive data for long-term access, consider the annotation of datasets, and so on. Context-aware systems are typically not designed to maintain information for long-term access, making annotation and privacy less important concerns in most cases. Moreover, context-aware systems, unlike life-log systems, do not need to always run in the background of the device eliminating the issue of resource usage.

The remainder of the paper is organized as follows. We begin with a discussion of related works. Next, design considerations of using mobile phones for life logging will be identified. Then, the framework architecture and its associated data model will be described. Afterward, an implementation of this framework will be described. Subsequently, the implementation of the framework will be evaluated. Finally, we conclude this paper.

2 Related work

As has been described before, using a mobile phone as a life-log tool is not new. However, to our knowledge, there is no life-log tool with an open architecture available, which considers the described challenges of storing lifetime information. Furthermore, existing applications or tools are not flexible enough to enable users configure sensors. In this section, we first list approaches that provide a life-log or a similar tool. Life-logs are subset of context-aware applications, and thus, life-log applications on mobile phones might be assumed as context-aware applications. Therefore, we discuss context-aware approaches that were designed for mobile phones (or any other pervasive devices such as PDAs), and they sense and record contextual information in a continuous manner. These

[1] http://www.google.com/tv.

related works served as inspiration for the UbiqLog architecture described in this paper.

Nokia Photos, formerly known as Nokia Lifeblog [6], is a life-log tool designed for Nokia mobile phones. Inputs for this life-log tool are images and videos captured by the phone's camera, sent and received multi-media messages, sent and received text messages, user created text notes, blog posts, and recorded sounds. Lifeblog enables users to post their information to their blogs. It has a closed architecture, and there is no possibility to extend or configure its sensors.

MyLifeBits [7], a research project proposed by Microsoft, focuses on capturing most desktop activities of users and storing them digitally. Desktop activities include read articles, books, emails, presentations, pictures, home movies, video taped lectures, and voice recordings. This project can be considered one of the largest efforts toward designing a life-log system. In addition, it supports some ubiquitous sensors such as Sensecam [8], which is a body mounted camera.

Reality Mining [9] employs mobile phones for life logging. They sense and record users' location (using bluetooth beacons), social proximity (using the phone's bluetooth sensor), phone usage including application usage and dialed and received calls. Text messages and call contents are not included. It focuses on learning the social behavior of a group of users. Unlike Reality Mining, our focus is on a life-log framework that contains an application architecture and its associated data model. As has been noted before, UbiqLog was not designed for a specific purpose. In other words, our research does not focus on any specific life-log use cases such as analyzing user's social behavior. iRemember [10] uses a PDA to continuously record users' conversations in order to help them augmenting their memory at a later stage. It converts users' generated audio to the text and enables them to search and browse these texts. Audio requires more storage than text; therefore, due to a lack of local storage on PDAs, audio data is transferred to a large storage server.

MyExperience [5] is a data collection framework for mobile phones, which allows users to automatically log phone sensors data. The recorded data can be enriched through self-report surveys. MyExperience senses and records device usage, user's contextual information, and environmental sensors. It is open source, and it provides an interface for configuring and extending sensors. These are useful features for a life-log tool. However, MyExperience is not a life-log tool, because it does not support other requirements for life-logs such as long-term preservation, annotation, user privacy, and resource efficiency.

Pensieve [11] is another mobile phone-based approach for augmenting users' memory. It uses captured pictures and audio as an input for its dataset. Likewise, it provides manual and automatic annotation features for the recorded data.

Experience Explorer [12] is a recent life-log approach, which uses a mobile phone to sense and capture contextual information. It provides an open interface to communicate with third parties such as the Flickr photo service[2], to let users share media content. In contrast to Experience Explorer, UbiqLog does not provide any social networking features, since social networking requires an external server that hosts the users' information and go beyond the scope of a mobile phone. Instead, the focus when designing UbiqLog was to stay within the mobile device and to exclude other issues such as connecting to external servers. It is notable that there is a feature to transfer life-log information to another media, but this is an optional component and not a mandatory option. This will be explained in more detail in the following sections. ContextPhone [13] is a software platform designed to sense and record contextual information, and it runs on the Symbian OS of Nokia series 60 smartphones. It consists of interconnected modules that can sense and record contextual information, connect and communicate with external services, customize applications such as recent calls and launch background services such as status display.

Lu et al. [14] described that long-term continues sensing on mobile phones is a challenge, because of lack of resources. Jigsaw [14] contains a design and implementation of a continuous sensing application that balances the performance needs of the application and the resource demands of continues sensing on the phone. Jigsaw has been implemented for GPS, Accelerometer, and Microphone sensors. The Jigsaw itself is an engine that runs in the background of mobile devices. Two applications have been built based on Jigsaw, JigMe, which provides a log of users' daily activity (life-log), and GreeSaw that provides carbon footprint and caloric expenditure information. Similar to Jigsaw, our approach could be used as API or application.

CenceMe [15] is a Symbian based platform designed for N95 Nokia Mobile phones. It is designed to infer users' sensing presence (e.g. dancing in a party with friends) and share this presence through a social network such as Facebook. As sensor, it employs GPS, Microphone, captured picture, accelerometer, and Bluetooth. CenceMe contains a mobile application and a backend infrastructure on a server machine. The mobile application performs sensing, classifies raw sensor data in order to produce meaningful data, and finally, it presents users' presence directly on the phone and upload this information to the backend server.

Mobile Lifelogger [16] is designed to provide a digital memory assistance via life logging. It collects accelerometer and GPS sensor data, and it tackles the issue of large

[2] http://Flickr.com.

life-log dataset by indexing those data objects using an activity language. Their indexing approach supports easy retrieval of life-log segments representing past similar activities and automatic life-log segmentation. In general, the proposed framework is composed of three parts: *Mobile Client*, *Application Server*, and *Web Interface*. The Mobile Client will be installed as a mobile application on users' phone and handles the sensing. The Application Server is responsible for storing, pre-processing, modeling recorded data as an activity language and finally retrieving similar activities of the user. The Web Interface is a web application that enables users to browse and access their life-log information. In order to facilitate information retrieval add semantic to raw sensor information, we provide semantic enrichment that will be described in the "Annotation" section. Moreover, from sensor extension and configuration perspective, our approach is flexible.

There are some context-aware approaches such as ViTo [17], UbiFit [18], and PEIR [19], which are not considered as continuous sensing tools, but they employ mobile phones or PDAs to sense contextual information and record them. ViTo uses a PDA as a home entertainment remote control device that records usage duration of T.V, Music player, etc. It assists users to alter their behavior by embedding behavior change strategies into normal interaction with the device. UbiFit is another smartphone-based approach to record users' physical activities. It provides an esthetic representation of the physical activities to encourage users do exercises. PEIR (Personal Environment Impact Factor) uses GPS of mobile phones to collect users location changes, and by using a accelerometer sensor data and Hidden Markov Model-based activity classification, it determines users transportation mode.

3 Design considerations

Mobile phones (or in general pervasive devices) by their very nature are not designed to host large computing applications. There are some restrictions such as resource limitation, user interface, etc. in comparison to personal computers. Life-log tools are very privacy sensitive, and this research proposes to bring this tool on mobile phones, which are prone to loss or damage [20]. It is therefore important to take some design considerations into account when implementing a life-log application for mobile phones. These considerations constitute the foundation for the implementation of the UbiqLog framework.

3.1 Controversial history of life-logs and security

Since life logging promises recording every activities of individuals, life-logs are very privacy sensitive tools. They have a controversial history, for example, the DARPA lifelog project [21] that was canceled in 2004 because of the criticism of the privacy implications of the system. Allen [22] identified two important potential hazards of life-logs: pernicious memory and pernicious surveillance. In another classification [23], risks have been listed as surveillance, long-term availability of personal information, the theft of life-log information, and memory hazards as potential risks of sharing life-log data with society. However, life-logs propose a complimentary assistance to our biological memory, and they are highly capable of augmenting our life. Therefore, these risks should not hinder technological development, and they can be reduced by considering users' security and privacy in the implementation of a life-log system. UbiqLog framework handles these issues via introducing a security component, which secures the sensing and recording process. This component will be described in more detail in the "Security and Privacy related issues" section.

3.2 User intervention

Sensors are the core components of a life-log. They continuously sense contextual information, and therefore, interruption and resumption of interruption should be addressed [24]. In particular, a service is always running in the background. Ideally a life-log application runs in the background of a device 24/7, hence user administration or user intervention should be reduced as much as possible. On the other hand, users should have appropriate understanding and enough control over the application. While users should be able to configure sensors, but sensing and logging (recording) processes should not depended on any user intervention. According to the expectancy theory [25], performing administration tasks has a negative impact on the user experience. UbiqLog does not require any user intervention or supervision during the sensing and recording phases, which is assumed to improve usability. Only administration tasks, such as configuring sensors or managing temporary disk spaces, require user intervention. Users have sufficient control on managing UbiqLog, but it does not interrupt users, and thus, it is unobtrusive, which is assumed to be an intelligibility fact. To evaluate the UbiqLog user interface, we performed an evaluation based on Nielsen's usability heuristics. Test result indicates that participants were satisfied with the user control on the application. Further details about the usability evaluation are provided in the "Evaluation and Verification" section.

3.3 Application performance

Mobile phones, like other pervasive devices, suffer from resource weaknesses [26]. Client thickness [20] of mobile applications, from resource usage perspective, is a

challenging issue in mobile computing. As described above, Life-log applications, by their very nature, are always running in the background of the system. A service running in the background of a mobile system should not consume too much resources, and it should not affect other functionalities of the system. It is therefore important to consider the resource usage of such applications. We study and analyze the resource usage and performance of this framework in the "Performance Evaluation" section. There we prove that running UbiqLog always in the background does not have significant effect on the target device resources.

3.4 Storage

The disk space available on mobile phones and other pervasive devices is smaller than on personal computers. While sensors that store data in the textual format do not consume much disk space, binary data such as audio, picture, or video consume more disk space. Satyanarayann [26] described that mobile phones, like other portable computers, are more vulnerable to loss or damage than desktops (reliability treat). On the other hand, as far as a life-log application is working, its dataset size is continuously increasing. All these facts lead us to conclude that a server with enough storage capacity must be the main residence of the data, and life-log information will be maintained only temporarily on the target phone. Data needs to be uploaded to the server either manually or automatically. Users should be able to access and manipulate life-log dataset on the server too, but uploading to an external storage is not in the scope of this research. Similar solutions were used in related projects, for example, iRemember [10] transfers recorded audio to a large capacity server.

3.5 Sensor extendability

Another potential problem with using smartphone sensors are their limited precision and weakness compared to external sensors. For instance, an external GPS device is more powerful than the phone's built-in GPS, or the quality of a photo from a digital camera is better than the quality of a photo taken with the phone's camera. On the other hand, mobile phones are increasingly equipped with more capabilities and features. It can be anticipated that in the future, new sensors will get integrated into mobile phones. Therefore, it is a crucial requirement to provide an open interface that allows adding new sensors. In order to deal with this problem, we propose an integration interface to enable developers integrate external sensors into UbiqLog. For instance, an external GPS can be connected to the phone via bluetooth. The interface for the external sensor integration will be discussed in more detail in the "Framework Architecture" and "Implementation" sections.

3.6 Support for multiple use cases

As described earlier, the information recorded by life-logs can be used in many different domains. Our focus is to provide a generic life-log framework, which is capable to be configured based on the users' requirements. Users can add/remove (extendability) or disable/enable (flexibility) sensors and change settings of available sensors. This capability makes the framework flexible enough to be used for different purposes. For instance, once it can be used as health monitoring device and in another use case it can be used to study a group behavior such as employees' geographical location changes in a firm. Li et al. [27] stated that an individual's life has different facets and available systems are mostly unifaceted. However, we claim that UbiqLog is a multifaceted system, because it is capable to be used for different use cases.

4 Data model

UbiqLog stores each life event as a data entity. A life-log dataset composed from a set of infinite life events. Each life event of users is a record in this dataset. We are living in a spatio-temporal world. Meaning, all of our life events except dreams happen in a specific location and at a specific date-time. Based on the current available technologies, it is not always possible to sense the location, because location sensors such as GPS do not function in every environment. For instance, a GPS can not function in indoor environments. There are other approaches such as A-GPS (Assisted GPS) to solve this problem, but they are not always able to sense location, and they are imprecise. On the other hand, most operating systems have date-time, which is accessible as long as the target device has not been turned off. This means most devices with computing capabilities can provide time-stamps. Therefore, we conclude that date-time is a necessary field for any life-log record, and all life-log information objects will be stored with the time-stamp.

The Life-log dataset of a user U represented via $L(U)$ and $L(U) = \{E_1, E_2, \ldots E_n\}, \quad n \to +\infty$. n is going to infinity because the user's life is ongoing, and the life-log dataset size is increasing continuously. The life-log data entity E is a 3-tuple (D, T, A), where T is the time-stamp, which can be continuous or discrete. If it is continuous, it will be the start time-stamp and the end time-stamp; if it is discrete, then it will be only a time-stamp. D is the information object, and it can be a binary data, for example,

image, audio or textual data, for example, GPS location, micro blog content, etc. *A* is the annotation associated with this data object. Annotations are text data and used to enrich the semantic of a data entity. More on this will be explained in the "Annotation" section. The UbiqLog framework was implemented based on this data model, and to our knowledge, yet there is no specific data model designed for life-log systems. Some example records of the proposed data model will be shown in the next sections. This data model is technology independent and can be implemented for any life-log system.

5 Framework architecture

In order to achieve a versatile and flexible architecture, we use a layered architecture model [28]. Components were designed to be abstract, and thus, it is possible to add new components or remove an existing component. One of the main design requirements was to create a flexible and extendable system, which enables users to integrate new sensors or configure current sensors. This sensor extendibility feature is called versatility in context-aware approaches [29]. Figure 2 shows the architecture of the framework. As it has been stated before, the proposed architecture is not intended for context-aware applications, although life-logs might be interpreted as a subset of context-aware applications.

Figure 1 shows the sensing process sequence diagram of this framework. It is notable that our focus was on creating a tool for data collection and not reflecting data to users. However, in order to enable end users evaluate the implementation, some visualization approaches have been proposed. We will explain them more in the "Usability Evaluation" section.

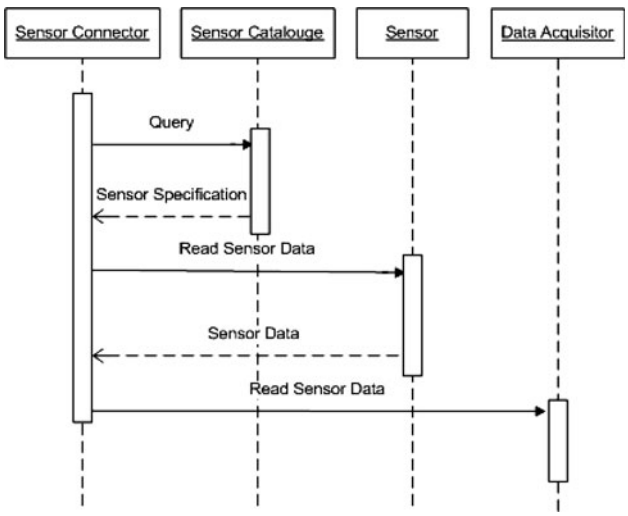

Fig. 1 Sensing process sequence diagram

In this section, we first describe general components, followed by a discussion of a set of components that are being used in the sensing and data collecting phases of the life logging process. *Application Log* is being used to log errors and messages of the application in order to assist developers in debugging. *Extension Interface* is used as an interface for adding new sensors to the framework. An external sensor might be an application, or it might be a hardware component such as a device with a Bluetooth connectivity feature. How to add a new sensor to the framework will be described in more detail in the implementation section.

Similar to other context-aware approaches, this framework contains two major operations, sensing and recording. Chen and Kotz suggested [30] to decouple the context sensing part from other parts of the application in order to maintain a flexible environment for sensors configuration. The UbiqLog framework is based on their suggestion. Mostly life-log tools are composed of two types of hardware: a data acquisition device and sensors. Sensors are responsible for sensing the environment and reading the contextual data in a raw format. The data format varies based on the sensor. The raw data of each sensor should be aggregated in a format that is appropriate to gather all readings in one dataset. Therefore, the data acquisition device is required, which is responsible to gather and aggregate raw data from the sensors. Aggregation here means converting raw data into a data format, which the framework is able to parse. In the UbiqLog framework, we use the mobile phones' sensors although it is possible to

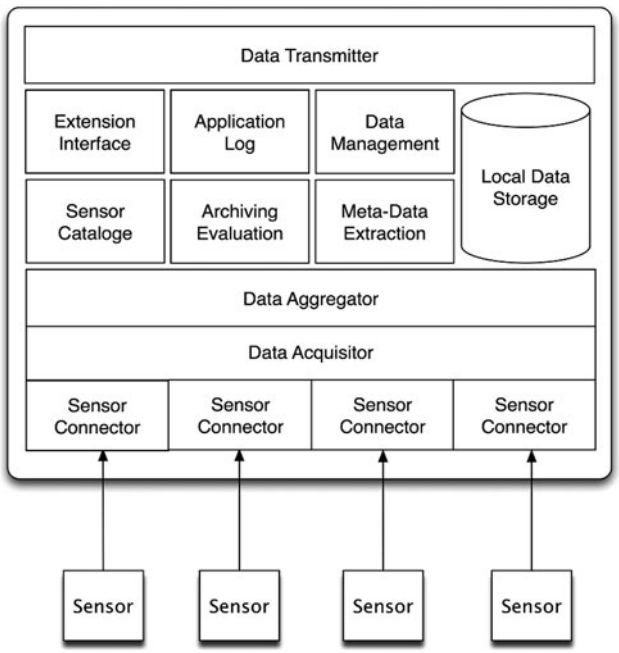

Fig. 2 UbiqLog framework architecture

integrate external sensor. This means that, in the implementation of this framework, sensors do not reside separately from the phone and their data will be stored on users' mobile phones.

Sensing and recording are assumed to be two different phases of the life-logging process. Following is the detail explanation about the sensing and the recording processes.

5.1 Sensing

Context data acquisition methods predefine the architectural style of the system. There are different approaches to read contextual data; in this framework, sensor data will be read directly via a sensor connector, and no middleware will be used.

The sensing process is composed of three component: *Sensors*, *Sensor Connectors*, and *Data Acquisitor*. "Sensors" are used to sense and read contextual information. They can reside in smartphones as an application or hardware component. They can also reside physically outside the smartphone, such as an external GPS device connected to the smartphone via Bluetooth, or a Web service, which provides temperature based on current location of the user.

The "Sensor Connector" establishes a connection between the framework and the sensor. The Connection depends on the sensor, for example, a sensor can send data via a Web Service or a sensor can write data in the file only. A security component might also be used by the sensor connector, especially when the sensor is physically located outside the phone. A network connection will be established between the sensor and the framework; hence an authentication or an authorization process might be necessary. In simple terms, the sensor connector encapsulates the sensor structure and reads data from the sensor. We suggest having a sensor connector for each sensor, as a sensor failure would otherwise affect other functionalities of the framework.

Information about sensors, their configuration, and their connection status will be kept in the *Sensor Catalog*. The sensor connector connects to the sensor and reads information based on the configurations, which reside in the sensor catalog. Table 1 shows default installed sensors and their configuration values in the implemented application of the UbiqLog framework. Users can list sensors from the application GUI as shown in Fig. 3 Frame 3. The Accelerometer is disabled by default, because it consumes high amount of resources and thus drains the battery. Other sensors are disabled, based on the result of our user survey, in which users stated which sensors have higher priority (enabled sensors).

The "Data Acquisitor" is a stream, which acts as a temporary queue to hold the read data from sensors.

Table 1 Default sensor configurations

Sensor	Configuration values
Application	Enable = yes, record interval in ms = 10,000
Telephone	Enable = yes, record communication = no
SMS	Enable = yes
Location	Enable = yes, update rate in ms = 10,000
Accelerometer	Enable = no, rate = delay_game, force threshold = 900
Temperature	Enable = no, measurement unit = Celsius
Compass	Enable = no
Bluetooth	Enable = no, scan interval in ms = 60,000
Orientation	Enable = no

It contains raw sensor data and is located between the data aggregator and sensor connectors. It encapsulates data that has been read from sensors in order to make it available for the data aggregator. This layer exists because it is not possible to send raw data from the sensor connector to the data aggregator directly, while there is no guarantee for reading sensor data synchronously in the real time; therefore, this component is required. For instance, an authentication process might take time, which means a data from that sensor cannot be read in real time. In addition, data aggregation processing cannot be done in real time. Thus, the data acquisitor will be used as a temporary sensor data holder (data buffer). This leads us to conclude that the acquired data from sensors will be logged passively and not actively.

It is notable that this architecture does not comply with any sensor classification unlike other context-aware efforts. For instance, Raento et al. [13] has classified smartphone's sensors as location, user interaction, communication behavior, and physical environment sensors. Froehlich et al. [5] categorizes phone's sensor as hardware sensors and software sensors such as application usage. Schmidt et al. classified sensors as logical and physical [31]. We do not provide any sensor categorization, to avoid that a categorization might restrict the extendability feature of the framework, since it is possible that in the future, new sensors are becoming available, which do not refer to any of the proposed categories.

As has been described previously, mobile sensors are less precise than external sensors. Likewise, it might be possible that in some situations, sensors are not available (resource can not be discovered). These problems should not affect other sensors. We handle this via reading sensors in parallel. In more technical terms, each sensor has an associated background service (in the implementation Android services were used). If a sensor is not available, it does not raise any errors, and it does not affect other reading processes. Only an error log entry will be created

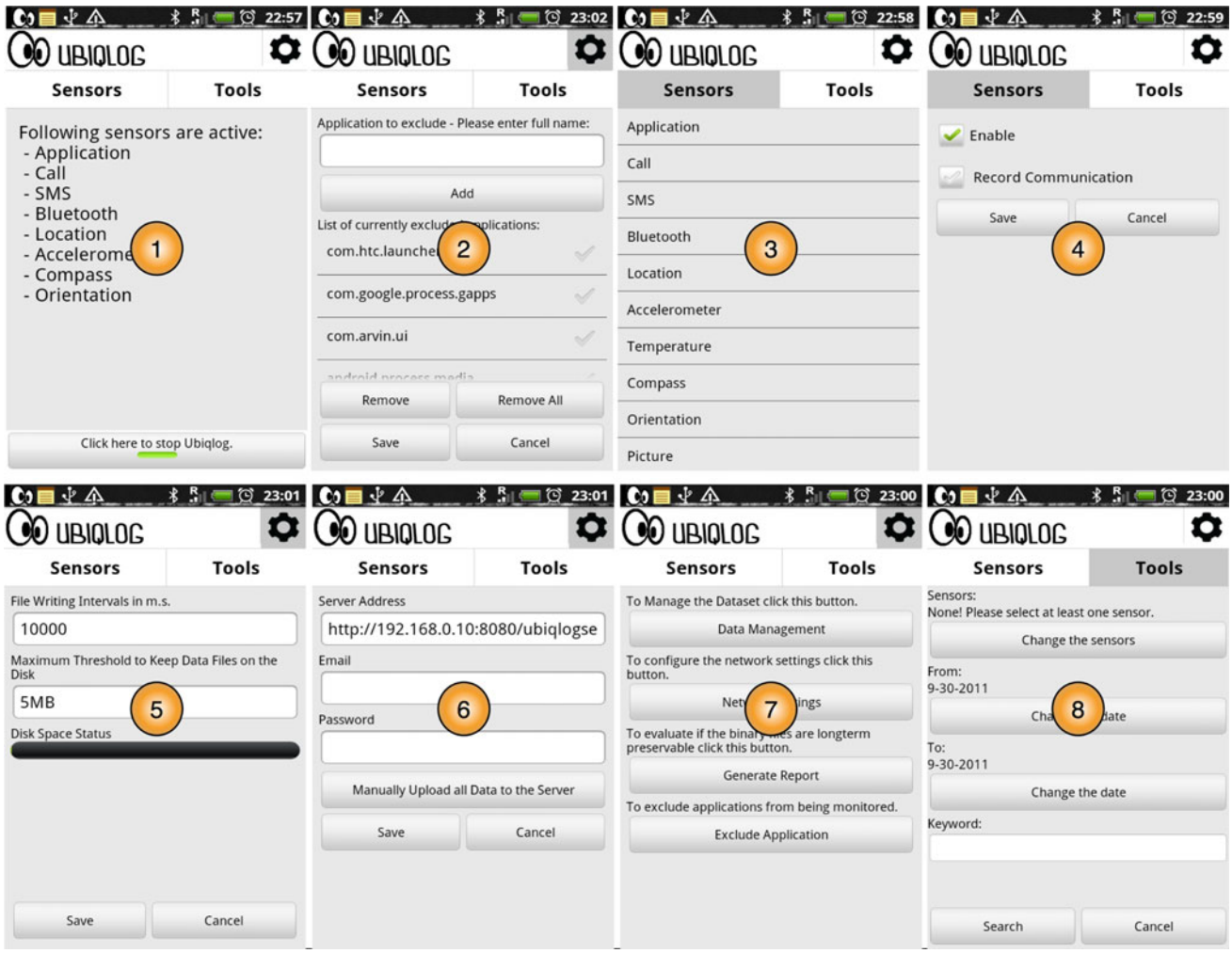

Fig. 3 GUI screenshots. *Frame 1* shows the first screen that appears when the user opens the application. The feature shown in *Frame 2* enables the user exclude specific application from being logged. *Frame 3* shows the list of default sensors, which were implemented in the application. Clicking on one of the sensors from this list takes the user to the configuration screen, shown in *Frame 4*. *Frame 5* shows the data management setting screen. *Frame 6* shows the screen for configuring network settings. *Frame 7* shows the setting options. *Frame 8* shows the search facility

for this sensor, and the associated sensor connector does not add any data to the data acquisitor.

Sensors and the mobile device are connected mostly in a unidirectional way. If an authentication or authorization mechanism is required, connection can be bidirectional between the mobile phone and the sensor.

5.2 Refining and recording data

The refining and recording phase consists of five components: *Data Aggregator*, *Metadata Extraction*, *Data Management*, *Local Data Storage*, and *Network Transmitter*. This phase considers scalability, manageability, and reliability of a life-log dataset.

Raw data from different sensors needs to comply with a consistent data format. Furthermore, the data needs to get annotated in order to facilitate browsing, search, and further data retrieval. The "Data Aggregator" receives raw data from "Data Acquisitor" and transforms data to a consistent structure, which is readable for the framework. In particular, Data Aggregator is a middle layer, which enriches sensor data via annotation and converts its format. Most other context-aware approaches [32] use XML for their data format. In the implementation of this framework, we convert data into the JSON[3] format, because a JSON document consumes less disk space in comparison to similar XML documents while being as flexible as XML. The following shows two examples of the aggregated data, one from the SMS sensor and the other one from the Application sensor:

[3] http://www.json.org.

```
"Application":
{"ProcessName":"com.android.browser",
 "Time":"Oct 15, 2009 6:21:40 AM"}

"SMS":
{"Address":"4444444", "Type:send",
 "Time":"Dec 24, 2009 11:23:01 PM",
 "Body":"test message" }
```

These JSON data entities comply with the described data model, because each entity has a date-time, a data object and an annotation. Binary data has the same structure, but instead of text, they hold the location of that binary data entity. For instance, a picture data entity is as follows:

```
CAMERA:{
"picturename":"2009-12-29 20.42.05.jpg",
"local-Location":"/sdcard/dcim/camera",
"Time":"Dec 29, 2009 8:42:07 PM"}
```

Converting a raw data to the desired data format will use the information from the sensor catalog, for example, find and use the related annotation class. Metadata Extraction component in the proposed architecture (Fig. 2) will be used to annotate data in order to make them searchable and browsable. Annotation here is adding a textual data to the JSON record. For instance, an annotated data entity from the call sensor might be as follows:

```
"Call":{"Number:11111111",
"Duration":"13", "Time":"Dec 24, 2009 9:23:04 PM",
"Type":"outgoing",
metadata:{"name":"John Smith"}}
}
```

The "metadata" element of the above JSON record represents the annotation. The Data Aggregator uses the "Metadata Extraction" component to annotate data of the sensors. The annotation process is not only restricted to be done during the data aggregation phase, it can also be done during the data acquisition phase. For instance, in the implementation, calls and SMSs get annotated during the sensor read, because by one query to the Android content provider this information can be read.

After the data getting annotated and converted to the data entity format, the resulting information object will be stored locally on the phone (data will be stored locally, because of the data connection cost and reliability of mobile phones. However, we enable users to transfer them to a reliable storage). These information objects will be written to the "Local Data Storage" by the Data Aggregator. The write process will be done passively and not actively or real time, because raw data must be converted to a readable data format. Furthermore, the data aggregation process cannot be done in a realtime, for example, an annotation could be done via an external web service call, which costs time. First data will be stored locally because of two reasons. A network connection is not always available, and previous research efforts [2] show that transferring data automatically to anther storage increases the risk of packet loss and requires data compression. Mobile phone storage has limited capacity; therefore, we provide a feature that allow users to manually upload the data to a reliable storage media (RSM) or even automatically. The RSM could be the personal computer of the device owner or a server on the cloud, which is capable of hosting life-log information. Uploading data to the RSM will be done by the "Network Transmitter" layer.

The life-log dataset consists of files, which are either binary files (pictures, videos, calls, etc) or text files. Manual upload to the RSM requires user intervention, which is not desirable. Therefore, "Data Manager" will be used. Data Manager is responsible to check whether the maximum size of the UbiqLog folder is reached or not; if it reaches the maximum size, a secure network connection will be established by the "Network Transmitter," and it uploads dataset files to the RSM. After a successful file receive, RSM acknowledges the framework. If files are successfully uploaded to the RSM, then Data Manager removes files from the local storage. Moreover, Data Manager is responsible to compress the content of text files. For instance, Data Manager can check the log file and remove redundant records or compressing files if there is lack of disk space on the local storage.

5.3 Annotation

Annotation describes the attachment of extra information to a piece of information [33]. Life-log datasets host large

amounts of contextual information about owners and their activities. In addition, dataset size increases continuously, and information retrieval is a major challenge. Since the data comes to the life-log dataset from heterogeneous sensors, basic issues that are being solved in RDBMS are critical challenges in the context of PIM and also life-log systems [34].

A possible approach for assigning semantics and enriching life-log datasets based on described requirements is the use of annotation. Here by annotation we mean tagging and thus metadata creation, which is data about the data. Metadata information objects will be extracted from other resources, and they will be collected based on date-time and (if possible) location. Annotation can be done either manually or automatically. Due to the fact that life-log dataset structure is always growing in size, manual annotation is not feasible and cumbersome. Annotation can be stored embedded in the information objects or separated from the target information objects [33].

We embed some annotations within the data entities in the "metadata" tag, as shown in the above examples. Some annotations will be stored separately from the life-log dataset, but those files have similar structure. Annotations that are not embedded in the data entities will be done by calling external web services. Those services collect textual annotation from other information resources such as personal calendar, social network activities, and news. In the implementation, we use a service to call Google calendar, and it collects calendar events in a text file. Although this feature is available, we suggest to perform this type of annotation on RSM and not on the device, because establishing a connection to an external service consumes bandwidth.

Moreover to consider the readability and flexibility of the framework, we suggest to perform the annotation during the data aggregation phase and not during the data acquisition phase.

5.4 Digital preservation

Digital preservation is maintaining information, in a correct and independently understandable form, over the long term [35]. Likewise, it can be interpreted as method to preserve the content in an understandable form in long term; therefore, digital preservation prevents data format obsolescence.

As has been described, life-log information is worth keeping during the owner's life. Bell et al. [4] noted that this problem is one of the major challenges of using life-log tools. There are two challenges with long-term archival of digital data, hardware, and software. The hardware problem is not in the scope of this research, but we provide a solution for the software problem.

A digital preservation process composed of different processes such as migration, evaluation, emulation, etc. [36]. Migration is a process of changing the format of a digital object to another format that is long-term archiveable [37]. Migration eliminates the need to retain the original application for opening binary files. Bell [38] named long-term preservable data formats "Golden" data formats. For instance, TIFF is a golden data format for long-term preservation of image files. Text files such as XML are golden data format. Life-log files are either in text format or binary format. Binary files format needs to be checked, and if it is not long-term preservable, it must be changed to a long-term preservable format.

Rawassizadeh and Tomitsch [39] proposed a framework to consider the long-term preservation of the digital objects for pervasive devices. There they suggested to handle the file conversion to an external Web service, because file conversion is a resource intensive process. We converted binary objects that are not long-term preservable to a preservable format. Therefore, users do not need to have any interaction with the system, and file conversion is being done in the background.

In the UbiqLog framework, we intend not to bound ourselves to any external tool such as Web services, which require external service call. Thus, users can choose how to handle their binary objects. Either they can use the described method or a second method, which is lighter. The second method is a simple evaluation mechanism based on the data format. If the file format is not preservable (not in a golden format category), an entry for the metadata information of that file will be created. All binary files in the framework have a record in a textual file, which contains information about the files. This textual file will be sent to the RSM with the associated binary files. File format conversion can then be handled on the RSM side, and the results can be automatically evaluated by another component. In simple terms, in the second method, only the file format will be checked on the mobile device. If it is not long-term preservable, an entry will be written in the text file. We suggest to use the second method, because it consumes less resources such as bandwidth and CPU, and there is no guarantee that external services are always available for the file format conversion.

5.5 Security and privacy

Allen [22] described that using life-logs with the existing privacy laws and policies, do not set appropriate limits on unwanted usage of information. She described that there is a high potential of incivility, emotional blackmail, exploitation, prosecution, and social control by government while using life-log tools. Strahilevitz [40] stated that the most private information consists of sensitive personal

matters such as sexual encounters and bodily functions, sensitive medical information and knowledge of owners fundamental weaknesses. A life-log tool can sense and record this information; therefore, from a privacy perspective, a life-log dataset is a very sensitive object. These facts show the importance of securing the life logging process. Security issues need to be considered during the design and implementation of life-log tools and not only after the implementation of the target tool is finished.

Usually a life logging system has three stages [41]. Each stage requires specific security considerations. The first stage is sensing the information from the user environment by sensors; the second stage is collecting the sensed information; and the third stage enables users to browse and retrieve information from their life-log dataset. Users should be able to define what information object they intend to collect. They might need to configure sensors in order to set their configuration parameters such as sensing interval, etc. The first stage connects the life-log system to sensors and reads sensors data. In this stage, two parts might require to be secure. First, some sensors might require authentication; second, if the sensors' data contains sensitive information, data transmission from the sensor to the life-log tool should be secure too, for example, encrypted data. The second stage collects the sensed information in the life-log device. This stage creates a dataset of the life-log information. The dataset contains a set of life-log records. Data that comes from sensors are mostly raw data, and in order to enable users to browse and access them, some changes have to be done on the raw data. Changes might include annotation, aggregating sensors' data, migrating data from one format to another format, etc. During the collection phase developers should consider the third party tools that they are using for changing data. For instance, a security threat might be the use of an annotation engine from a third party, which sends users' information to that third party.

The third stage is storing the life-log data. Here storages, which host life-log information, should be secure. We suggest maintaining data in an encrypted format if data is intended to be stored as text files, or if it will be stored in a database, designers should consider to define appropriate access limitation on the database.

This framework performs all of those three stages internally, and no connection to a third party tool will be established. Furthermore, we do not use any external libraries for these stages. Therefore, we respect security by those stages internally, without any external access. However, if a sensors require a third party tool or library for sensing, we can not guarantee the security of that specific sensor.

Usually life-log devices are pervasive devices such as mobile phone, and not desktop application. Unlike desktop computers, pervasive devices are prune to loss or damages [26]; hence they are not capable of hosting personal information. Therefore, life-logs should maintain their data on reliable storages such as the personal computer or cloud of the user. If a life-log tool does not use a pervasive device, then there is no need to have a local storage, and data can be stored directly on a reliable storage media. But life-logs usually contain at least a pervasive device. Transferring data from a pervasive device to a reliable storage media might not be a standard stage for a life-logging process, but usually it is required. Communications and data transfers are very sensitive from a security point of view. This demonstrates that during the communication, data should be transfer encrypted. Securing the connection can be done by the transport layer security (TLS). However, additional to TLS, using a digital signature has been suggested as an additional way to secure the data transportation. Here we secure the communication from the phone to the server. It means communication with the server is encrypted via implementing TLS (Transfer Layer Security) and using HTTPS instead of HTTP. *Security Component* represents security-related issues such as authentication and authorization.

6 Evaluation and verification

To evaluate the UbiqLog framework, first an implementation of the proposed architecture on the Android 2 platform will be described. The purpose of this implementation was to evaluate and improve the feasibility of the framework. Then as a longitudinal evaluation, a nine month usage of this tool on user's mobile phone will be analyzed. Since the life-log tool needs to be constantly running in the background of the device, it is important to evaluate the battery efficiency during use the implementation. Finally, a usability evaluation of the application based on Nielsen's heuristics [42] approaches will be described.

6.1 Implementation

We have developed an implementation of the proposed architecture on the Android 2.0, 2.1, and 2.2 platforms. The implemented application was used and tested on a Motorola Milestone(Droid), HTC Legend, Samsung Galaxy S, HTC Desire and HTC Desire HD. This application fully complies with the described architecture.

Figure 3 shows the graphical user interface (GUI) of the application. A major consideration during the GUI design was to provide sufficient functionality to enable users to easily configure sensors without any knowledge about the lower layer of the application. When users click on any sensor in the sensor list, as shown in Fig. 3, Frame

3, they will be shown the configuration screen for each sensor such as Frame 4, Figure 3. The user navigation flow between the different screens of the application is shown in Fig. 4. Each box represents a screen in the application. Each Sensor Reader has been implemented as a separate Android Service. Since Android services run in the background, no GUI is required to run these services. For each sensor, there should be an associated sensor reader class. In order to add a new sensor into the application, a developer must implement the "SensorConnector" java interface in the "com.ubiqlog.sensor" package. If developers intend to add the associated annotation class, they should create a class in the "com.ubiqlog.annotation" package, and this class has to implement the "Annotation" java interface. Currently, in the implemented application, we annotate incoming and outgoing calls and text messages (SMS) with the recipients or senders names, which are read from the contact list. After the developer creates the annotation class they need to rebuild the framework and installs the new apk file on the target device. This apk file contains newly added sensor. There is no possibility to add new sensors dynamically or during the run time to the framework, which means end users can not do them. They can only configure existing sensors and enable/disable them.

Android provides listener classes for some sensors; when the sensor senses a new data, the listener will be notified, for example, calls information can be read via using "PhoneStateListener" Listener. We use these listeners to log sensors' data, but for some sensors such as application usage, there is no such a listener. Those sensors will be checked frequently based on a specified time interval via their associated sensor readers.

As described above, the Sensor Catalog contains information about sensors. A table in the SQLLite database of the Android platform has been used for the Sensor Catalog implementation. This table contains sensor names, sensor reader class, sensor annotation class, and a list of configuration data for that sensor.

In order to implement the Data Acquisitor, which is a temporary data buffer, a static Java Collection object (ArrayList) was used. The sensor connectors write their data directly to this object.

Data Aggregator runs as an Android service, it reads data from the Data Acquisitor, and it performs the annotation.

Life-log data is stored as a set of text files on the SD card of the phone in a folder called "ubiqlog"; binary files such as photos can be stored in different location. Location and metadata about each binary file is stored in the same file, which will be created for every day the application is being used. Users can use the "Search" menu and access their life-log data. They can also manipulate the search result (modify or remove an entire record).

Life-log files will be uploaded to the RSM by the HttpClient package of the Android. Password and server address are required, as shown in Fig. 3 Frame 6, to allow the application connecting to the RSM and uploading files. On the RSM side, we have designed a simple Java Servlet, which reads files and stores them locally on the user's personal computer. After a successful file receive, the RSM acknowledges the application in the Http response (HTTP Status Code is 200)that the files have been successfully uploaded to the RSM. Afterward, the Data Manager removes the files from the SD card of the device. Users can specify the maximum size for the UbiqLog folder as shown in Fig. 3 Frame 5. The Data Manager can check whether the threshold

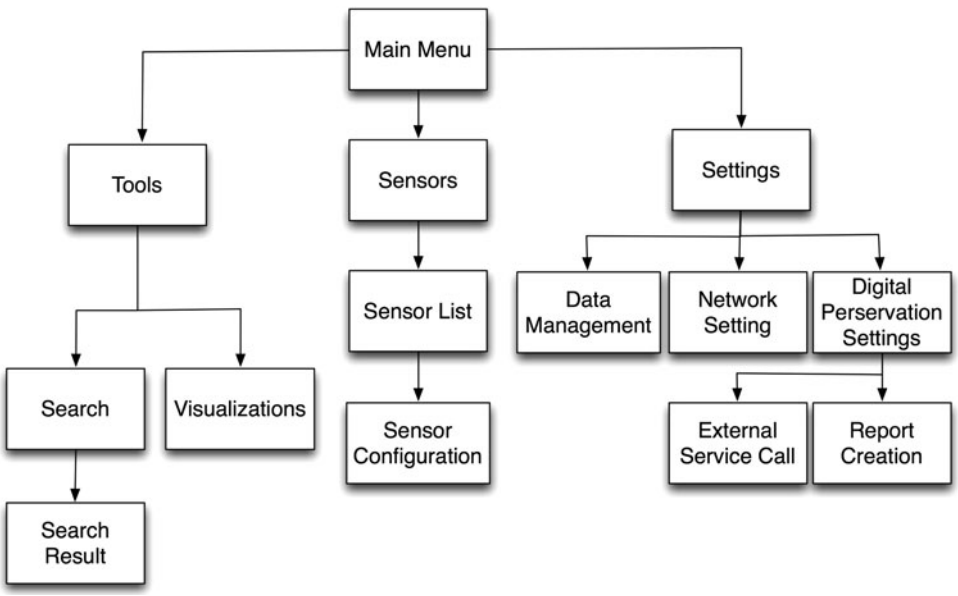

Fig. 4 GUI screens navigation flowchart

has been exceeded or not, and if the threshold has been exceeded, it will upload life-log files automatically to the RSM. The Data Manager has been implemented as an Android service, which continuously runs in the background, with the sensor readers and other services.

"Archiving Evaluation" is another Android service, which checks whether binary files are in a long-term archive-able format. As described above, this service can be used in two ways, either to call an external Web service to convert file format or mark the binary files that they are not in the long-term archive-able format by adding a corresponding metadata for this file in the associated record.

In order to check the long-term archive ability of life-log dataset binary files, users can manually generate the report by using the "Generate Report" button on the "Setting" Frame as shown in Fig. 3 Frame 7. Calling an external Web service to change the format of the file is another method that we described it in another paper [39].

6.2 Longitudinal analysis

The application described in the previous section was used by six users over a period of one to fourteen months. Three users immediately installed the application after they purchased the new phone. This means that they used the application from the first day of using their phone. Every day a text file was created containing sensor information. It contains location of the binary objects, and their metadata. Figure 6 shows the size of log files for about three months. Those logs are generated by using a Motorola Milestone phone. The reason why the file size was larger in the beginning of the study period was since the user described that he is keen in using a new phone and discovering the device features or surfing the market for new applications. This behavior is repeated by two other users with new phones. Therefore, we conclude this is the reason of having a large log files in the first days of usage. Besides, when users have lots of location changes, file sizes increases. Figure 6 shows two days in this period with zero file size. On those days, UbiqLog was not running; therefore, no file was created. The visualization shown in Fig. 5 demonstrates how the data recorded with the UbiqLog tool can be used to visualize the users social activity. This visualization approach was inspired by Song et al. [43]. Individuals' names have been pseudonymized in respect to the users' privacy; therefore, phone numbers are shown with Person and a number. The longer the call duration, the closer the person is to the center of the circle.

6.3 Resource usage evaluation

Jain [44] described that the application performance is composed of five factors: usability, throughput, resource usage, response time, and reliability. Evaluating the performance of an application is necessary for evaluating the implementation of the framework, since the UbiqLog application will always run in the background (and therefore always consume resources). As mentioned above, resource consumption is one of the major challenges of mobile computing [26], which is especially relevant for life-log application, since they typically need to constantly run in the background. Thus, the performance of the application can influence the general functionality of the device, which is not designed for this type of application usage. Therefore, resource usage monitoring is an important factor that developers need to consider while designing such applications.

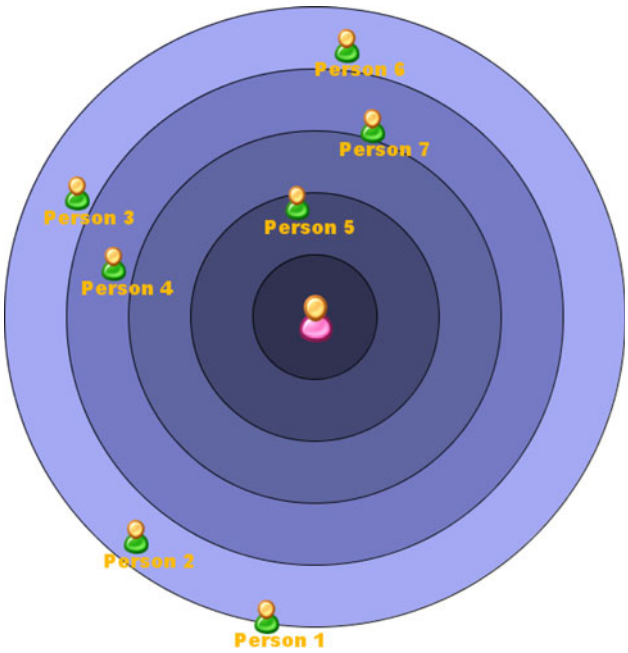

Fig. 5 Social activity of the user based on the duration of received and dialed calls

The application was tested directly on the device at the end of the implementation cycle to eliminate any data redundancy and sensor reader malfunction.

Mobile resources include battery usage, CPU usage, memory usage, disk activity, and network activity [45]. To measure mobile phone resource utilization, we used a resource monitoring tool [45]. We specifically measured battery, CPU, and memory usage of our application. Network activity and disk I/O of the life-log were not measured, since the application did not have heavy network or disk activity. Disk write happens infrequently because we buffer data and then write them in the file.

When all sensors were active, result of the CPU utilization monitoring showed that our application consumes less than 3 % of CPU in average, when administration activities are performed and the GUI is active, it consumed about 10 % of CPU in average. Average VmRSS (Virtual

Memory Resident Set Size) was 15728_Kb, and VMSize is 127268_Kb. It is notable that disabling some sensors (i.e. reducing the number of sensors) did not affect the CPU or memory utilization. This result shows that the implementation of the UbiqLog consumes fair amount of the CPU and the memory. Battery utilization cannot be measured per process. The GPS sensor reads the user's current location, and the Bluetooth sensor scans the environment for discovered devices, based on a configurable time interval. The default Bluetooth scan interval is 6 minutes, and the default GPS location read interval is 10 seconds. Both Bluetooth and GPS are highly battery consuming, but they produce important information for the life-log dataset. Therefore, we needed to investigate whether the implementation of the framework had any significant impact on battery consumption. In order to perform this study, we investigated battery utilization under different conditions, where different sensors were activated. Each test was started at a set time in the morning with a fully charged battery. Tests were run until the battery level reached a level of the 20 %. Device usage conditions have been kept constant during the entire test period. This means that the device was still used to answer phone calls or SMS, but not for any unusual purposes, such as playing games. However, there is no guarantee that the device usage is exactly the same for all tests, because the user can not control incoming calls or etc.

Table 2 shows the approximate time it took for the phone to reach a battery level of 20 % in different scenarios. It is notable that Wi-Fi, which consumes high amount of battery, was always on in all scenarios. Furthermore, to preserve more battery network connection of the phone was set to 2G (not 3G).

The intention of this study was not to evaluate battery usage of different hardware settings. Instead we intend to prove that the implementation of the proposed UbiqLog framework does not have a large impact on battery utilization. As shown in Table 2, there is about one hour difference between using the application and not using it while Bluetooth and GPS are disabled. With Bluetooth and GPS both enabled, the application decreased the battery time for half an hour. This might be due to I/O operation increase (writing GPS and Bluetooth log on SD Card)

6.4 Usability evaluation

This framework and its implemented prototype targets life-log system developers, but it can benefit end users by providing self-insight. In order to enable end users benefit from the framework, we propose following visualizations Fig. 7, which assist users in better self-awareness and self-monitoring about their device usages. In order to evaluate the usability of the framework implementation, we have

Table 2 Approximate time it took for the phone to reach a battery level of 20 % (starting from 100 %) in different scenarios. When WiFi and Bluetooth are on, they were idle and not active

Battery discharge duration (hours)	UbiqLog application	WiFi	GPS	Bluetooth
21:00	Deactive	On	Off	Off
20:00	Active	On	Off	Off
7:30	Active	On	On	Off
6:30	Active	On	On	On
7:00	Deactive	On	On	On
14:30	Active	On	Off	On

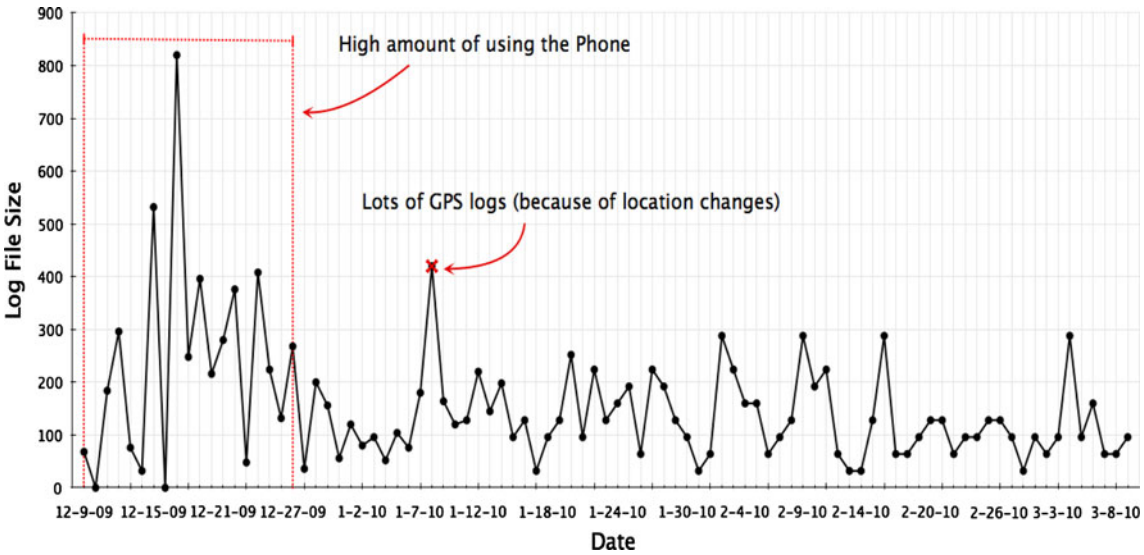

Fig. 6 File sizes of the life-log dataset. The unsteady part represents that user is playing with his new phone features(the area is marked with *red*). Other days when the file size is big mostly is because of using GPS outdoor and many location log entries have been generated (color figure online)

Fig. 7 *1* Location, *2* application usage, *3* call visualizations, and *4* movement

employed Nielsen's usability heuristics [42]. The implementation that users evaluate contains those visualizations under the "visualization" option of the "tools." The usability of the implementation has been evaluated by six users (2 female, 4 male) between 25 and 37 years of age. Four of the participants stated having strong computer knowledge; and other two stated having basic computer knowledge. All participants use computers and mobile phones in their daily life. Participants owned HTC Desire, Motorola Droid (MileStone), Samsung Galaxy S, HTC Legend, and HTC Desire HD. We installed the application on the participants' phones and asked them to use the application for a period of four weeks to one year. After this period, we conducted interviews and asked them to fill out a survey. The survey included Nielsen's principles for user interface design [42]. We adapted those principles in form of questions and asked participants to rank the application accordingly. The result of the evaluation shows *Help and Documentation* received the lowest score (2 from 5), but other heuristic factors received satisfactory scores (4 or 5 from 5).

To meet the design requirements, based on the result of those evalutions, "seamless integration into daily life" will be supported by unobtrusive and continues sensing. We have also described our approaches for the "Security," "information retrieval," and "long-term digital preservation." Moreover, the resource usage evaluation indicates the "resource efficiency" of the UbiqLog implementation.

7 Conclusion

To our knowledge, there is no open architecture available for life logging tools. Most scientific efforts focus on a specific use case of using life-logs without publishing any information regarding their architecture or data model. This makes it necessary to custom-build every new life logging application running on smartphones, which is expensive and time consuming, and requires advanced programing skills and systems architecture knowledge. Additionally, with life-logs becoming increasingly important as source of information for behavior learning, health monitoring, memory augmentation, etc., it is necessary to provide users with a flexible architecture and data model allowing them to adjust the application for their life logging needs.

To address these issues, we presented UbiqLog, a life-log framework, which is flexible and extendible to add new sensors and change the configuration of existing sensors. UbiqLog consists of an open architecture and propose a data model specifically designed for life logging. The generic approach of the architecture enables developers to implement it on any devices with computing capabilities, such as e-book readers, mobile phones, T.V.s, etc. UbiqLog is a generic and holistic framework, which can be used for different use cases and can be configured based on the user requirements.

To evaluate the proposed framework and data model, we developed an implementation of the UbiqLog framework on the Android 2 platform for smartphones. The implementation was evaluated regarding resource utilization, longitudinal analysis, and usability requirements.

Acknowledgments We would like to thank all students who helped us in developing this platform, including Victor Andrei Gugonatu and Soheil Khosravipour.

References

1. Ryoo D, Bae C (2007) Design of the wearable gadgets for life-log services based on UTC. IEEE Trans Consum Electron 53(4):1–6

2. Choudhury T, Borriello G, Consolvo S, Haehnel D, Harrison B, Hemingway B, Hightower J, Klasnja P, Koscher K, LaMarca A, Landay J, LeGrand L, Lester J, Rahimi A, Rea A, Wyatt D (2008) The mobile sensing platform: an embedded activity recognition system. IEEE Pervasive Comput 7(2):32–41
3. Roussos G, Marsh A, Maglavera S (2005) Enabling pervasive computing with smart phones. IEEE Pervasive Comput 4(2):20–27
4. Bell G, Gemmell J, Lueder R (2004) Challenges in using lifetime personal information stores. In: SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on research and development in information retrieval, p 1
5. Froehlich J, Chen M, Consolvo S, Harrison B, Landay J (2007) MyExperience: a system for in situ tracing and capturing of user feedback on mobile phones. In: 5th international conference on Mobile systems, applications and services (MobiSys '07), pp 57–70
6. Myka A (2005) Nokia lifeblog—towards a truly personal multimedia information system. In: Proceeding of workshop des GI-Arbeitkreises mobile datenbanken and informationsysteme
7. Gemmell J, Bell G, Lueder R (2006) MyLifeBits: a personal database for everything. Commun ACM 49(1):88–95
8. Hodges S, Williams L, Berry E, Izadi S, Srinivasan J, Butler A, Smyth G, Kapur N, Wood K (2006) SenseCam: a retrospective memory aid
9. Eagle N, Pentland A (2006) Reality mining: sensing complex social systems. Pers Ubiquit Comput 10(4):255–268
10. Vemuri S, Schmandt C, Bender W (2006) iRemember: a personal, long-term memory prosthesis. In: CARPE '06: Proceedings of the 3rd ACM workshop on continuous archival and retrieval of personal experiences, pp 65–74
11. Aizenbud-Reshef N, Belinsky E, Jacovi M, Laufer D, Soroka V (2008) Pensieve: augmenting human memory. In: CHI '08: CHI '08 extended abstracts on Human factors in computing systems, pp 3231–3236
12. Belimpasakis P, Roimela K, You Y (2009) Experience explorer: a life-logging platform based on mobile context collection. In: Third international conference on next generation mobile applications, services and technologies, pp 77–82
13. Raento M, Oulasvirta A, Petit R, Toivonen H (2005) ContextPhone: a prototyping platform for context-aware mobile applications. IEEE Pervasive Comput 4(2):51–59
14. Lu H, Yang J, Liu Z, Lane N, Choudhury T, Campbell A (2010) The Jigsaw continuous sensing engine for mobile phone applications. In: Proceedings of the 8th ACM conference on embedded networked sensor systems (SensSys 2010), pp 71–84
15. Miluzzo E, Lane N, Fodor K, Peterson R, Lu H, Musolesi M, Eisenman S, Zheng X, Campbell A (2008) Sensing meets mobile social networks: the design, implementation and evaluation of the CenceMe application. In: Proceedings of the 6th ACM conference on embedded network sensor systems (SenSys'08), pp 337–350
16. Chennuru S, Chen P, Zhu J, Zhang Y (2010) Mobile Lifelogger-recording, indexing, and understanding a mobile user's life. In: The second international conference on mobile computing, applications, and services (MobiCase 2010)
17. Nawyn J, Intille S, Larson K (2006) Embedding behavior modification strategies into a consumer electronic device: a case study. In: 8th international conference on ubiquitous computing (UbiComp 2006), pp 297–314
18. Consolvo S, McDonald D, Toscos T, Chen M, Froehlich J, Harrison B, Klasnja P, LaMarca A, LeGrand L, Libby R, Smith I, Landay J (2008) Activity sensing in the wild: a field trial of ubifit garden. In: CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems, pp 1797–1806
19. Mun M, Reddy S, Shilton K, Yau N, Burke J, Estrin D, Hansen M, Howard E, West R, Boda P (2009) Peir, the personal environmental impact report, as a platform for participatory sensing systems research. In: Proceedings of the 7th international conference on mobile systems, applications, and services (MobiSys 2009), pp 55–68
20. Satyanarayanan M (2001) Pervasive computing: vision and challenges. Pers Commun IEEE 8(4):10–17
21. Shachtman N (2003) A spy machine of DARPA's dreams. http://www.wired.com/print/techbiz/media/news/2003/05/58909. Last Accessed 6 Aug 2010
22. Allen A (2008) Dredging up the past: lifelogging, memory, and surveillance. Univ Chic Law Rev 75(1):47–74
23. Rawassizadeh R (2012) Toward sharing life-log information with society. Behav Inf Technol. doi:10.1080/0144929X.2010.510208
24. Abowd G, Mynatt E (2000) Charting past, present, and future research in ubiquitous computing. ACM Trans Comput-Human Interact (TOCHI) 7(1):29–58
25. Vroom V, MacCrimmon K (1968) Toward a stochastic model of managerial careers. Adm Sci Q 13(1):26–46
26. Satyanarayanan M (1996) Fundamental challenges in mobile computing. In: Fifteenth annual ACM symposium on principles of distributed computing (PODC '96), pp 1–7
27. Li I, Dey A, Forlizzi J (2010) A stage-based model of personal informatics systems. In: CHI '10: Proceedings of the SIGCHI conference on human factors in computing systems, pp 557–566
28. Wac K, Pawar P, Broens T, van Beijnum B, van Halteren A (2010) Using SOC in development of context-aware systems: domain-model approach. In: Sheng M, Yu J, Dustdar S (eds) Enabling context-aware web services: methods, architectures, and technologies. Chapman and Hall/CRC, pp 171–210
29. Lugmayr A, Saarinen T, Tournut JP (2006) The Digital Aura—Ambient Mobile Computer Systems. In: 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP '06), pp. 348–354
30. Chen G, Kotz D (2000) A survey of context-aware mobile computing research. Tech. rep
31. Schmidt A, Aidoo K, Takaluoma A, Tuomela U, Van Laerhoven K, Van de Velde W (1999) Advanced interaction in context. In: First international symposium on handheld and ubiquitous computing, pp 89–101
32. Baldauf M, Dustdar S, Rosenberg F (2007) A survey on context-aware systems. Int J Ad Hoc Ubiquitous Comput 2(4):263–277
33. Hunter J (2009) Collaborative semantic tagging and annotation systems. Annu Rev Inform Sci Technol Am Soc Inform Sci 43(1):187–239
34. Jones W (2007) Personal information management. Annu Rev Inform Sci Technol 41(1):110–111
35. Consultative Committee for Space Data Systems (2001) CCSDS 650.0-B-1. Recommendation for space data system standards: reference model for an open archival information system (oais). Tech. rep
36. Waugh A, Wilkinson R, Hills B, Dell'oro J (2000) Preserving digital information forever. In: Fifth ACM conference on digital libraries (DL '00), pp 175–184
37. The Commission on Preservation and Access and the Research Libraries Group, Washington, D.C. : Preserving digital information : report of the Task Force on Archiving of Digital Information (1996)
38. Bell G (2001) A personal digital store. Commun ACM 44:86–91
39. Rawassizadeh R, Tomitsch M Towards digital preservation of pervasive device information. http://www.personalinformatics.org/docs/chi2010/rawassizadeh_digital_preservation.pdf (2010)
40. Strahilevitz L (2005) A social networks theory of privacy. Univ Chic Law Rev 72(3):919–988

41. Rawassizadeh R, Tjoa A (2010) Securing shareable life-logs. In: IEEE international conference on privacy, security, risk and trust, first international workshop on privacy aspects of social web and cloud computing (PASWeb-2010), pp 1105–1110
42. Nielsen J (1994) Ten usability heuristics. http://www.useit.com/papers/heuristic/heuristic_list.html
43. Song M, Lee W, J, K (2010) Extraction and visualization of implicit social relations on social networking services. In: Twenty-fourth AAAI conference on artificial intelligence (AAAI 10), pp 1425–1430
44. Jain R (1991) The art of computer system performance analysis: techniques for experimental design, measurement, simulation and modeling. Wiley, New Jersey
45. Rawassizadeh R (2009) Mobile application benchmarking based on the resource usage monitoring. Int J Mob Comput Multimedia Commun 1(4):64–75