Exploring mobile health in a private online social network

Qurban A. Memon* and Asma Fayes Mustafa

EE Department,
College of Engineering,
UAE University,
Al-Ain 15551, UAE
Email: qurban.memon@uaeu.ac.ae
Email: 201180954@uaeu.ac.ae

Abstract: Health information is very vulnerable. Certain individuals or corporate organisations will continue to steal it similar to bank account data once data is on wireless channels. Once health information is part of a social network, corresponding privacy issues also surface. Insufficiently trained employees at hospitals that pay less attention to creating a privacy-aware culture will suffer loss when mobile devices containing health information are lost, stolen or sniffed. In this work, a social network system is explored as a m-health system from a privacy perspective. A model is developed within a framework of data-driven privacy and implemented on Android operating system. In order to check feasibility of the proposed model, a prototype application is developed on Facebook for different services, including: i) sharing user location; ii) showing nearby friends; iii) calculating and sharing distance moved, and calories burned; iv) calculating, tracking and sharing user heart rate; etc.

Keywords: social networks; privacy; role-based access; healthcare network; healthcare social network; m-health.


Biographical notes: Qurban A. Memon has contributed at levels of teaching, research and community service in the area of Electrical and Computer Engineering. Currently, he is working as an Associate Professor of Electrical Engineering at the UAE University, College of Engineering. He has authored/co-authored over 80 publications in his academic career. He has executed research grants and development projects in the area of microcontroller-based systems; network intelligence and security. He has served as a Reviewer of many international journals and conferences; as well as session Chair at various conferences. His research interests include intelligent systems and networks.

Asma Fayes Mustafa completed her Bachelor’s degree in Computer Science in 2010 from the Abu Dhabi University, UAE, and later Master’s degree in Electrical Engineering from the College of Engineering, UAE University, 2014. Her research interests are in social networks and corresponding applications.
1 Introduction

Wireless sensor networks (WSNs) have become mature to be adopted and deployed at commercial scale. WSNs could be made more appealing to end users by leveraging online social networks (OSNs) using developed novel applications and interaction cases between WSNs and OSNs. Nowadays sensors and social networks can fruitfully interface, from sensors providing contextual information in context-aware and personalised social applications, to using social networks as ‘storage infrastructures’ for sensor information. The integration of sensor networks with social networks leads to applications that can sense the context of a user in better ways and thus provide more personalised and detailed solutions. Social networks have gained immense popularity recently with the advent of sites such as MySpace, Friendster, Facebook, etc. These networks are a source of data as users populate their sites with personal information. To better understand how OSNs can be integrated with physical world, there is a need to understand services provided by current OSNs, which are:

- identity and authorisation services
- application programming interfaces (APIs) to access and manipulate social network graph, publish and receive updates
- container facilities for hosting third party applications, as discussed by Blackstock et al. (2011).

There are various important drivers for integrating sensor and social networks. One driver for integrating sensors and social networks is to allow the actors in the social network to both publish their data and subscribe to each other’s data either directly or indirectly after discovery of useful information from such data. The idea is that such collaborative sharing on a social network can increase real-time awareness of different users about each other. The second driver for integrating sensors and social networks is to better understand or measure the aggregate behaviour of self-selected communities or the external environment in which these communities function. Aggarwal and Abdelzaher (2011) exemplify cases such as understanding traffic conditions in a city, understanding environmental pollution levels, or measuring obesity trends.

Some integration examples may be quoted, such as the Google Latitude application, which shares the collected mobile position data of the user. In this paper, the proximity alerts may be triggered when two linked users are within geographical proximity of one another. As another example, the City Sense application collects sensor data extracted from fixed sensors, GPS-enabled cell phones and cabs in order to determine where the people are, and then carries this information to clients who subscribe to this information. A number of real-time tracking applications such as ‘automotive tracking application’ determine the important points of congestion in the city by pooling GPS data from the vehicles in the city. Animal tracking uses tracking data collected with the use of radio-frequency identifiers (Aggarwal and Abdelzaher, 2011). Miluzzo et al. (2007) discuss how to inject sensing presence into popular social networking applications such as Facebook, MySpace and IM (Skype, Pidgin) to allow new levels of ‘connection’ and implicit communication between friends in social networks. The Green GPS is a participatory sensing navigation service that maps fuel consumption on city streets, to allow drivers to find the most fuel-efficient routes for their vehicles between arbitrary
end-points (Ganti et al., 2010). The Microsoft SensorMap allows for a general framework where users can choose to publish any kind of sensor data. The SensorMap application enables users to index and cache data. The indexing and caching allows users to issue spatio-temporal queries on shared data (Aggarwal and Abdelzaher, 2011).

Sensors provide numerous research challenges. Since the collected data may contain sensitive personal information (e.g., location data), it is extremely important to use privacy-sensitive techniques to perform analysis. Another challenge is that the volume of data collected can be very large. For example, in a mobile application, one may track the location information of millions of users simultaneously. The innovations in World Wide Web as discussed in Memon and Khoja (2009, 2010) and the recent trends in data protection as explored in Moravejosharieh et al. (2012) and Memon (2006) have increased attraction for use of OSNs. However, related advancements in technology and tools are also complemented by corresponding privacy concerns. Altshuler et al. (2012) highlight such concern and describe the lack of awareness for potential risks as the most important thing when data are shared online. As an example, the external entities can mine this data and use it for different purposes like spamming (Huber et al., 2010), discover interaction pattern in the enterprise to offer and develop innovative services, identify the important person in the network, detect hidden clusters, identify user sentiments for proactive strategies (Fung et al., 2010).

Data security and information privacy have surfaced to be of a major concern in many areas including healthcare specially in the area of Electronic-Health (e-Health), and Mobile-Health (m-Health) since emerging mobile applications for data transport relate to telemedicine. This data includes transmitting health-related information and research. As such other health-related activities such as cardiovascular monitoring and patient location tracking can also be monitored using such applications. Corresponding research in e-Health and m-health issues and deployment challenges has also gained pace. As an example, Escarfullet et al. (2012) aim at developing a m-health system by logically linking characteristics of the m-health system together based on information flows. The respective authors use a questionnaire to develop a conceptual data model for the m-health system, and then break down the m-health data information flows to determine features to be focused on. The authors conclude that developing an efficient, user-friendly m-health feature is crucial to the success of development of an m-health system. In another work, Sun et al. (2012) provide a foundation for developing security solutions for organisations’ digital information and contribute to higher security in the medical information systems. Specifically, the authors present a purpose-based access control suited for e-Health system. In usage control, not only decision factors, such as authorisations, obligations and conditions are analysed, but also the continuity properties. However, the respective authors also agree that much work is still to be done before these models can be used in practice. A similar work proposed in Gupta et al. (2014) presents a rigorous definition of proximity based on formal topological relations, and shows that it can be applied to social networks, communication channels, etc. For this, the authors develop a formal model of proximity that is generic enough to specify all types of proximity to enforce security.

In terms of privacy enforcement, a lot of research work related to social networking and case studies in healthcare data privacy can be found in literature. For example, Squicciarini et al. (2014) discusses an approach that helps users in OSN users to manage their contacts into social groups by leveraging the social group practice to automate
privacy setting process. In order to implement this, the respective authors propose an incremental social group maintenance algorithm to set privacy settings for newly added content or new contacts joining their social circles. In another work (Park et al., 2014), the authors analyse the vulnerabilities in existing OSN services to develop a framework to provide trusted data management in OSN services. Specifically, an approach is proposed to determine optimum levels of information sharing taking into consideration the payoffs based on the Markov decision process. In order to address interactions of users with attackers with conflicting interests, the authors propose a game theoretic approach to allow a large amount of variables to be changed to suit particular setups. Example case studies related to healthcare can also be found. Chourasia and Tiwari (2012) develop a foetal e-health monitoring system using mobile phones and wireless sensors for providing advanced healthcare services in rural areas to avoid unnecessary visits to healthcare centres. For this purpose, the authors develop biotelemetry system for foetal e-health monitoring involving mobile phone with the help of a clinical staff. The security of transmission is enforced using a user-selected encryption key. Likewise, Adibi (2012) evaluates the patient acceptance feasibility to use mobile phones to collect, transfer, and receive health related data and instructions to assist certain diabetes patients. The authors propose the information in various layers and levels in the BlackBerry Enterprise Solution, which includes the deployment of National Security Agency (NSA’s) Suite-B Cryptographic algorithms.

As a summary, current fitness bands, electronic health records, health information exchanges, connected devices, tools, and sites containing medical data keep growing. Attackers are also becoming more sophisticated. Cybercriminals are seeking more information than ever about their victims to sell. The concern is that richer personal identity data of individual users, consisting regional and geographic data, personal information and behaviour are expected to be traded in the same manner that stolen credit cards are today. Thus, the task of securing health information is becoming more challenging for even the best-prepared organisations.

The purpose of this research is to extend social networking in the area of healthcare by sharing specific information (for example accumulated fitness indicators) online with doctor and/or parents in a privacy aware manner. The paper is structured as follows. In Section 2, a model is proposed that describes what is needed to build such an application, and which social parameters need to be linked, and how the platform addresses privacy. Section 3 presents experimental work, specially a prototype application development along with results. In Section 4, integration of this application to social networks other than Facebook is discussed. The comparative analysis is carried out in Section 5, followed by conclusions in Section 6.

2 Proposed model

A convenient design gives rise to other challenges that need to be addressed in order to enable successful mobile application development which meets user needs. Before describing the model, it seems necessary to narrate problem statement and highlight objectives of this work.
2.1 Problem statement

The mobile devices have very good computational efficiency, but they offer limited programming and resource usage control. Moreover, the devices have energy limitations. For example, GPS only works outdoors, and it quickly consumes phone battery power, thus application developers on mobile device platforms need to be aware of power consumption when an application is developed that depends on using radio interfaces such as GPS. Android’s Network Location Provider helps to determine user location using Wi-Fi signals and cell tower. This strategy helps to provide location details indoors and outdoors and uses less battery power. For effective use of battery, the application needs to deploy both strategies. Such a platform has privacy issues. The collected data such as location data contains sensitive and personal data. It is highly important to apply a privacy model to maintain data security. The data collected is huge, so it is extremely important to efficiently process huge amounts of sensitive data. Additionally, the application should be easy to understand and thus target non-expert.

2.2 Objectives of this work

The objectives of this work include extending OSNs by adding sensing features such as heart rate monitoring, sugar levels etc., and then processing and clustering some of this information for a benefit to the user in an environment like healthcare, using a locally developed application in the mobile device. The model should present an application that enables members of a typical social network platform to share healthcare related features along with their location information with their friends in a private manner. The application is expected to allow new levels of connection and communication between contacts in social networks. In return as a benefit, the services may motivate the user to compete with friends who exercise more and burn more calories. The application is expected to encourage users to track their heart rates and share it with their doctor. As a result, this is expected to help the user to track fitness to a new level.

Based on the stated objectives, a conceptual model is developed and displayed, as shown in Figure 1. The platform, components, and services that are presented in the model are detailed as follows.

A. Application platform: There are several popular OSN platforms. The Facebook is the most popular OSN platform today. The application is developed for such a platform and is compatible with Android devices. For development, the Facebook supports different APIs for developers:

i The Graph API, which is a simple HTTP-based API that gives access to the Facebook social graph, uniformly representing objects in the graph and the connections between them. Most other APIs are based on the Graph API.

ii The Open Graph API allows applications to tell stories through a structured, strongly typed API.

iii The Facebook offers a number of dialogs for Facebook Login, posting to a person’s timeline or sending requests.
iv The Facebook query language (FQL) enables the developer to use a SQL-style interface to query the data exposed by the Graph API. It provides some advanced features not available in the Graph API such as using the results of one query in another.

v The Facebook public feed API lets the developer read the stream of public comments as they are posted to Facebook.

Figure 1 Conceptual model (see online version for colours)

B. Application sensors: The application uses the built-in GPS of the user mobile device to get current location coordinates. It also uses the temperature and humidity sensor to check the weather temperature.

C. Application services: The services, which can be used using this platform, are:

i **Show nearby places**: This application enables the user to use the built-in GPS or Androids’ network location provider built in the mobile device to get the current location. It displays a list of nearby places as well.

ii **Public location badge**: The user can post location directly on Facebook to increase visibility of information to other users.

iii **Show nearby contacts**: The user gets his contact location based on last Check-in. The user can see if any of his or her friends are checked-in nearby. The application displays a list of nearby friends, place and the time they checked in.
iv  *Cluster nearby contacts*: Clustering is important in analysis and exploration of data. This application clusters nearby friends into groups based on colleagues, family and so on.

v  *Tracking distance moved, calories burned and active time*: This application tracks distance moved, calories burned and shows active time for the user. The application can also be used for running, cycling, walking and all other distance-based outdoor sports. Once data are shown on network, the user can seek extra encouragement from friends and family to workout. The clinical staff from a healthcare centre, once connected in privacy mode, can also monitor shared clinical data.

vi  *Get weather temperature*: This application enables the user to use the built-in sensors of a mobile device to check the weather temperature and share with friends.

vii  *Share pictures*: The application enables posting image of user’s specific injured or monitored body part, once on line, to be seen and examined by a doctor.

2.3  *Model data flow*

Based on the model just presented, the information and process flow inside application can be easily visualised. Based on this, the software architecture is shown in Figure 2. For the purpose of implementation, the various stages in information and process flow are discussed below.

*Figure 2  Software architecture of application (see online version for colours)*

A.  *Sensing*: Most Android-powered devices have built-in sensors that measure motion, orientation, and various environmental conditions. These sensors provide raw data with precision and accuracy. The platform supports three broad categories of sensors:

i  Position sensors to measure the physical position of a device. This category includes orientation sensors and magnetometers.

ii  Environmental sensors to measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity. This category includes barometers, photometers, and thermometers.

iii  Motion sensors to measure acceleration and rotational forces along with three axes.
This category includes accelerometers, gravity sensors, gyroscopes and rotational vector sensors. In this application, position and environmental sensors are used.

B. Data acquisition: Data acquisition is the process of gathering information in an automated fashion from analogue and digital measurement sources such as sensors. Apart from position and environmental sensors to get position (i.e., latitude and longitude information) and weather information, the application uses Facebook APIs such as Graph API and FQL to extract the user education history, user work history, friend’s last check-in coordinates, friends education history, friends work history and the nearby places.

C. Data processing: The application analyses and processes the extracted data to produce meaningful information. After getting the user location coordinates and his or her friends’ location coordinates, the application measures the distance between the user and each one of his friends to produce list of nearby friends. The application compares the user education-history and other details with friends’ education-history to cluster the nearby friends into groups such as colleagues, family and so on. Additionally, the application uses the acquired location coordinates taken frequently to measure the distance walked, the related duration and the calories burned.

D. Data sharing: The application enables the user to share location, the distance walked, burned calories, the weather temperature and pictures. The Graph API updating is done simply with an HTTP POST request to relevant endpoint with the updated parameters. To publish and share new data, the application uses POSTs HTTP requests to appropriate URLs.

E. Presentation: After processing the data, the application displays a list of nearby places. The user will be able to check in to any of these places by clicking on one of the places. It also has nearby friends icon. By pressing on this icon, the user will get a list of his or her nearby friends based on their last check-in. The application organises nearby friends into groups of colleagues, work friends, family and others. Moreover, it displays the distance the user walked, total calories burned and the weather temperature.

2.4 Privacy model

As highlighted in Section 1, data privacy is a major concern. In order to maintain privacy, the user can turn off this whole or components of this application using the settings option, but this is not the common practice of online users. In order to protect users’ shared data, data can be handled in many ways:

i. By bifurcating table into shareable and non-shareable columns, i.e., data partitioning, and then play with rights. This will work only in the case when we exactly know which field to be shared and your table entries are fixed (i.e., not dynamic). However for data porting and scaling up the options, one may face problems.

ii. By using MongoDB. MongoDB is a no-SQL database and works on documents rather than entries. It’s also hierarchical and supports dynamic data (i.e., table entries cannot be fixed).
iii (Jugar option) by inserting one column of rights to all your data rows. In the first part of query, check is done to see whether rights are correct, then further processing of query is allowed, otherwise the query is rejected.

iv By using role-based access control (RBAC) mechanism. In RBAC case studies, permissions are associated with roles, and users are to be made members of appropriate roles (Memon, 2013; Memon et al., 2007).

This helps to simplify management of permissions. Roles are similar to the group’s concept in access control. Role is defined as a set of users on one side and a set of permissions that will be applied to the users on the other side, while groups are defined as a set of users. The privacy model made as part of the proposed scheme is derived from this RBAC.

In the proposed model, a society represents social network with four properties: contexts, for example the number of users \((U)\); tasks, for example services such as displaying online friends; resources available to users; and policies that are enforced on all users, for example adult information, etc. Likewise, a community represents a user created and terminated dynamic group with five properties: contexts such as purpose of the community; tasks for example community creation; resources such as data sharing; cooperation such as collaborative cooperation \((CR)\); and community role. Similarly, a user has four properties: contexts that represent user status such as gender; tasks such as posting messages; resources created by user such as photos; and policies of two types: control policy to select users from amongst user list; and filtering policy to assign friends such as close or not close. A society, community or user has permissions \((Perms)\) through permissions assignment.

Thus, roles are generated for various trust levels and contacts are assigned roles based on their relationship with the user. Contacts can be easily reassigned from one role to another. The developed application has predefined role-permission relationships, which make it simple to assign contacts to the predefined roles. It is difficult, without the new privacy model, to determine what permissions have been assigned to what users. The proposed privacy model helps to perform large-scale authorisation management. The basic concept of the proposed privacy model is that the user assigns his contacts to roles, roles have predefined permissions, and contacts acquire permissions by being members of roles. User-role can be many to many, which means that the same user can be assigned to many roles.

The application provides the user with three role categories:

- trusted
- semi-trusted
- un-trusted.

The user will set the members of each role according to his or her relationships with his or her contacts. Trusted role category has predefined permissions that will enable its members to view and comment on most of the user posts. Semi-trusted will enable its members to view and comment on some of the user posts, while un-trusted role category members would not be able to view or comment on many of the posts. Moreover, the user can assign relationships to his trusted and semi-trusted contacts. The user can assign his
trusted contacts close or not close relationship. Assigning close relationship to contacts will give them the option to view more posts, while not-close relationship would not give them the permission to view more details. Using this privacy model, the users can allow their contacts to share certain level of their information. This model helps in managing different level of information sharing. Contacts will not know what role or relationship the user has assigned them. Such a model is shown in Figure 3.

**Figure 3** User-data-driven privacy model in a social network (see online version for colours)

It is clear from Figure 3 that each of the contacts has some level of access to the user data. For example, the contacts 1, 3 and 8 (i.e., friend, colleague, office staff) may only access name of the user, while contact 2 (a healthcare unit staff or user-parent) may also access address, location and medical records though all contacts may be part of same social network. In order to examine different relationships between entities, consider set of users, roles, objects, operations, collaborative relationships, access levels, and conditions be represented by \( U, R, Obj_s, Opr_s, CR, AL, Cond \), respectively. Thus, the assignment relations among elements of the privacy-aware model are:

\( 2^R \) : the power set operations

- Many to many mapping user-role assignment relation: \( URA \subseteq U \times R \).
- The set of permissions: \( Perms = 2^{(Opr_s \times Obj_s)} \). This can also be stated as:
  \[
  Perms = \{ (Obj_s, Opr_s) \mid Obj_s \in Obj_s, Opr_s \in Opr_s \}.
  \]
- The set of sharing and privacy-aware permissions:
  \[
  PA\_perms = (Perms, CR, AL, Cond)
  \]

This can also be stated as:

\[
PA\_perms = \Big\{(perms, cr, al, cond) \mid \begin{array}{l}
perms \in Perms, \cr cr \in CR, al \in AL, and cond \in Cond \end{array}\Big\}
\]

- Many to many mapping permission-role assignment relation: \( PRA \subseteq Perms \times R \).

Many to many mapping sharing and privacy-aware permission-role assignment relation: \( Prv \_PRA \subseteq PA\_perms \times R \).
3 Experimental work

Before results are presented, it seems necessary to highlight development environment, briefly stated as follows:

Simulation environment: The application is developed for Android operating system devices. Android is an open source operating system available to all developers with various expertise levels. Android is a Linux-based operating system created for touchscreen mobile phones and tablets. Android platform allows users to develop, install and use their applications. The application is primarily developed in Java programming language by using the Android software development kit (SDK). The SDK provides the developers with all the tools they need including software libraries, debugger, sample code, emulator, and tutorials. The integrated development environment (IDE) for developed application is Eclipse using the Android development tools (ADT) plugin. Facebook SDK for Android was used to integrate this application with Facebook’s platform.

In order to start the simulation, prerequisites including Eclipse, Android SDK, android developer tools (ADT) Plugin, and Facebook SDK were installed. Then, Facebook SDK was imported to Eclipse. Each Android app that is developed was signed, as it was required to register each application’s key hash with Facebook as a security check for authenticity. After registering on Facebook Developer Site as a developer, application Facebook profile was created and details such as application name, category, and key hash for were entered. After creating application profile, application ID was generated and appeared in the profile. Then application ID was added to project files on Eclipse.

In the following paragraph, the main focus is laid on how functions, components and services mentioned in Section 2 are implemented. The sample code of some of the developed functions/components is shown in Appendix.

a Login: This has been implemented in an easy way for people to log in to the application. The application uses iOS, Android, JavaScript and Facebook SDKs to speed up the process and build login systems quickly. For secure authorisation Facebook uses the OAuth2.0 open protocol for confirming a person’s identity and giving them control over right of access to their information.

b Permissions: The permissions enable developers to request access to information about someone using their application. It asks for the following permissions: offline access, publish stream, publish check-ins, photo upload, user status, user education history, user work history, friends’ status, friends’ education history and friends’ work history. To gain access, the application requests the permissions transparently through the Login dialog. To maintain information security, almost all API calls at Facebook need to have an access token passed in the parameters of the request.

c Show nearby places: The following steps outline how to get user’s current location, display a list of nearby places and check into one of these places with the Facebook SDK for Android.

i Set up the place picker item: This step includes defining a BaseListElement class to represent an item in the list. This class contains member variables that define the user interface (UI) as well as methods that are sub-classed to
implement the behaviour around click events, storing and restoring state info, as well as notifying observers about data changes.

ii Show the places picker: The Facebook SDK provides a placePickerFragment class that displays a list of nearby places. This fragment is hosted in the PickerActivity class. This activity launches when the user clicks on a place in the list. The PlacePickerFragment is used if the incoming intent data matches a pre-defined place picker Uri. Before loading the data, the PlacePickerFragment is configured to specify search criteria such as radius, query and maximum results to return.

iii Display the selected place: In this step, the place will be displayed when the place picker activity is dismissed.

d Public location badge: The following steps outline how to publish a story to share the user location with friends. A request will be published by using Request(Session session, String graphPath, Bundle parameters, HttpMethod httpMethod). GraphObject and OpenGraphAction interfaces are used to set up a Graph object representation of the POST parameters. Facebook SDK is used to publish the user location by performing the following steps:

i Construct a new Request for currently active session that is an HTTP POST to the me/checkins Graph API path.

ii Set a GraphObject for the Request instance. The GraphObject represents location parameters, such as the selected place ID, message and location coordinates.

iii For best practices, the user is asked for publish_actions write permission in context, when the app is about to publish the user location.

e Show nearby contacts: To show nearby friends, FQL is used. FQL enables to use a SQL-style interface to query the data exposed by the Graph API. Below, the steps are described that show nearby friends.

i Issue a HTTP GET request to /fql?q=query where query is a JSON-encoded dictionary of queries. The developed code uses FQL to get the friends details and location according to their last check-in, and is shown in Appendix 1.

ii Store the friends’ details and locations from JSONObject into the following arrays: latitude, longitude, author_uid_array, timestamp and checkin_id. After that, the distance between the user and each one of his or her friends is calculated and stored in distances array. The developed code (shown in Appendix 2) uses the response for the FQL query from the previous step and extracts the friends’ details and locations from the response, and then it stores them in arrays. After that the distance is calculated and stored in distances array.

iii Find out nearby friends by comparing distance between the user and each one of his or her friends with a predefined distance, then store nearby friends’ name in an array. The developed code (shown in Appendix 3) uses the distances array from the previous step to compare the distance between the user and each one of his or her friends with a predefined distance-threshold in order to determine nearby friends.
iv Display nearby contact names, their exact location and when they checked in. For example: Dr Noor Checked in Al-Ain Hospital at 2014-02-10 T09:16. The code written for this (and shown in Appendix 4) shows a list of nearby friends with their location, when user checks-in. A sample view of the list is shown in Figure 4.

Figure 4  Nearby friends list (see online version for colours)

f  Cluster nearby contacts: The following steps show, how clustering can be used to group nearby friends.

i  Issue a HTTP GET request to /fql?q=query to get user academics history.

ii Issue a HTTP GET request to /fql?q=query to get nearby colleagues after placing the user education history in the FQL query and display the nearby colleagues for the user.

g  Tracking distance moved, calories burned and active time: The following steps show, how tracking distance moved, calories burned and active time is calculated.

i  The application tracks the user moved distance by capturing user location coordinates periodically, calculating the distance between each two locations and summing the distances from the time the user presses start button till the time the user presses stop button.

ii The application will request the user to enter his or her weight in kilograms in order to calculate the walking burned calories. It uses the equation as shown below to calculate rate of calories burned per pound of body weight, as suggested by Karkanen (2000). A sample code for this is shown in Appendix 5.
Rate per Pound (Cal/lb-min) = A + BV + CV² + KDV³

where
V = Walking Speed (mph) – Limited to a minimum of 1 mph and a maximum of 5 mph
A = 0.0195
B = −0.00436
C = 0.00245
D = [0.000801(W/154)0.425]/W
W = Weight (lbs)
K = 0 or 1 (0 = Treadmill; 1 = Outdoors)

When the user presses stop button, the application displays the distance walked, the duration spent during the walk and the walking burned calories. Figure 5 shows resulting display on the mobile device.

Figure 5  The application workouts (see online version for colours)
h Get weather temperature: The following steps show how weather information is collected.

i To acquire data from temperature and humidity sensor, an instance of the SensorManager class is created. This instance is used to get the physical sensor.

ii Register a sensor listener in the onResume() method, and start handling incoming sensor data in the onSensorChanged() callback method.

iii Implement onAccuracyChanged() and onSensorChanged() callback methods. The sensor is unregistered when an activity pauses to prevent the sensor from continually sensing data and draining the battery.

i Share pictures: In order to share pictures, the application issues a HTTP POST request to share a photo with friends or healthcare professionals.

j Heart rate monitor: Heart rate monitor uses the camera and its flash to find the user heart rate in beats per minute. The user has to hold the tip of his or her index finger over the camera lens of his or her phone. The application takes between 10–30 s to get an accurate heart rate. Heart rate monitoring is based on using the camera with as little focus as possible. When the user puts his or her finger on the camera lens, it would not be focused. The resulting image will be only shades of light and dark RGB. The code looks at single channel (red) and tries to find out when the channel goes from light to dark red.

The application uses the ‘PreviewCallback’ mechanism to capture the last image from the preview frame. Then the YUV420SP data is processed to get all the red pixel values. Data smoothing in an integer array is used to figure out the red pixel average value in the image. The heart beat is detected when the average red pixel value in the latest image is greater than the smoothed average. The application collects data during 10 seconds, and then adds the beats per minute to an integer array which will be used to smooth the beats per minute data. As an illustration, Figure 6 is the resulting display of the user heart rate using this application. Together with it is shown for comparison purposes, a reading using another medical device at the same time. It turns out that the results are similar.

k Implementing privacy model: In order to ensure privacy, the application provides the user with three roles: trusted, semi-trusted, and un-trusted as shown in Figure 7. Predefined permissions are assigned to each one of the roles. The members of each role will be assigned by the user from his contact list.

a When the user presses Pick Trusted Friends, the following steps will occur:

i Issue a HTTP GET request, shown below, to /fql?q=query to get user friends list as shown in Figure 8. A sample code is shown in Appendix 6.

b When the user presses Add Friends, after choosing the ones he wants to select as trusted, the following steps will occur:

i Issue a HTTP POST request to /fql?q=query to create Trusted Friend list.

ii Issue a HTTP GET request, shown below, to /fql?q=query to get Trusted Friend list id, then, a POST request to add the selected friends to the list.
The same process will be executed when the user presses Pick Semi-Trusted Friends or Pick Un-Trusted Friends. When the user presses Assign Trusted Relationships, the user will get the layout as shown in Figure 9. If he or she pressed ‘Close Friends’ or ‘Not-Close Friends’, he or she will get a list of the trusted friends only to choose the close or not close friends from them.

c When the user presses ‘Close Friends’, the following steps will occur:

1. Issue a HTTP GET request to /fql?q=query to get user trusted friends only as shown in Figure 10. The sample code for this is shown in Appendix 7.

d Adding Friends to close or not-close will be performed the same way as mentioned in the trusted friend list.

**Figure 6** User shared heart rate (see online version for colours)
Figure 7  Privacy in the application (see online version for colours)

Figure 8  User friend list (see online version for colours)
Each time a post will be shared such as ‘distance moved’, ‘calories burned’, ‘active time’, ‘weather temperature’, ‘heart rate or pictures’, a number of steps are to be executed to specify who can see and comment on the post and who can’t. As an implementation, a sample code is shown in Appendix 8.

Figure 9  Assigning trusted relationship

![Close Friends](image)

Figure 10  Trusted friend list (see online version for colours)

![Add Friends](image)

4 Integrating with other social networks

There are various ways to integrate Android apps with social networks such as Facebook, Twitter and LinkedIn. One of them is using the SDKs available from Facebook, Twitter, etc. which provide the API to share updates. The second is using an embedded browser control and use OAuth for authentication and finally the REST APIs provided to post the update. These two ways require downloading and using the different APIs or implementing the complete protocol. The third and the best way is to use an open source
SDK which is available and easy to use. It provides integrating with several social networks. It is known as SocialAuth Android SDK. It allows posting status updates, getting user profile from Twitter, Facebook and LinkedIn using the API.

First the developer has to register the developed application with the social provider and get the API keys and secrets. Then SDK is integrated, which contains the java libraries that perform the heavy lifting of OAuth as well as the REST calls for social providers. The SDK provides the developer with multiple ways to show the user interface for selecting the social providers on which the user may want to share updates (Code Project, 2014).

5 Comparative analysis

A number of recent applications designed in context of integrating WSNs with OSNs can be examined for the purpose of comparison. The existing platform applications such as Google Latitude share the collected mobile position data of the user among different users, and then it generates proximity alerts when two linked users are within geographical proximity of one another. These applications are limited and target specific service only. In this work, the application not only uses the built-in sensors of the user mobile device (i.e., to get current location coordinates, view user current location, share location, show nearby places, and show nearby friends), but also provides more services such as clustering nearby friends, tracking distance moved, calories burned and active time, heart rate monitoring, getting weather temperature and sharing pictures. This may help to improve healthcare awareness and prompt quicker and safe advice from healthcare professional in social network. In this work, the data privacy is enforced by both options: the first option is available on all available social network platforms – as ON/OFF, while the second option is using user assigned roles vs. data levels.

This work can also be compared with Chourasia and Tiwari (2012), where foetal e-health system is developed to monitor data at local places instead of the clinic. Comparatively, our work has more advantages as the monitored data is accessible to larger concerned entities through a private social network. Similarly, the work by Adibi (2012) targets security of patient data transmitted through mobile phones using Blackberry platform and integrates whole patient information system, where as our work targets data in a private social network and is thus flexible from user perspective.

6 Conclusions and future work

In this paper, the framework and implementation of a m-health application was presented. A number of sensors were used to build a sensing application that enables members of a social network to share their information with their contacts in a private manner. The user can see if any of his or her contacts are checked-in nearby. It was shown that contacts can be clustered based on colleagues, family or any other criterion. The clustering process takes place in the user device. In terms of benefits, the application is a great tool for fitness, weight loss, calorie counting, etc., and can facilitate quicker monitoring of, for example, heartbeat of the user through social network by concerned healthcare units. Social constraints such as privacy were addressed in two ways: simple feature like turn
application ON or OFF; and the other by privacy aware data connectivity based on user roles.

There are a number of design challenges and obstacles to consider this as a complete m-health system. The issues relate to the hospital domain. We only considered the doctor as an entity, when we started building up the system where as the doctor is in fact a component in a hospital information system. Pertaining to a complete m-health system, there are a number of issues to considered:

- What is the role of the doctor in this private social network seen from hospital information system?
- How the user is allowed to enter any doctor in his or her social network?
- Once this data is viewed by a doctor, how this data is made as a part of the patient information system?
- If the doctor has a further comment/suggestion or observes a serious condition of the patient in a private social network, how this situation is further handled, for example, through ambulatory services?
- Since data travels through wireless domain, how is the patient data protected?

In order to make it a complete m-health system, above issues need to be addressed. In addition to the above, some additional features as a future work can be performed in order to extend and improve the built-application. Some of its aspects can be improved and more functionality added. Below are some suggestions for further improvement:

- Health support for elders by using sensor information to send alerts if there is abnormal activity. Request for attention can be sent to doctors and nearby friends based on the collected information from sensors such as body position and health measurements.
- Suggesting nearby friends based on common interests.
- Adding more features such as monitoring and tracking user blood pressure. This also includes storing, analysing and sharing the user blood pressure measurements.

References


Appendices

Appendix 1: Friends list and location (see online version for colours)

Bundle params = new Bundle();
params.putString("method", "fql.query");
params.putString("query", "SELECT author_uid,timestamp,coords,checkin_id FROM checkin
WHERE author_uid IN (SELECT uid2 FROM friend WHERE uid1 = me()) ");
String response = UtilityordovaFacebookRequest(params);
response = "{"data":" + response + "}";

Appendix 2: Store the friends’ details and locations

JSONObject json = Utilordova.parse(json( response ));
JSONArray data = json.getJSONArray( "data" );
JSONObject coords;
Long author_uid=(long);
for ( int i = 0, size = data.length(); i < size, i++ ){
JSONObject friend = data.getJSONObject(i);
if(author_uid!=friend.getLong("author_uid"))
{
    coords = data.getJSONObject(i).getJSONObject("coords");
    latitude[counter] = coords.getDouble("latitude");
    longitude[counter] = coords.getDouble("longitude");
    author_uid = friend.getLong("author_uid");
    author_uid_array[counter]=friend.getLong("author_uid");
    timestamp[counter] = friend.getString("timestamp");
    checkin_id[counter]=friend.getLong("checkin_id");
    loc.distanceBetween(loc.getLatitude(), loc.getLongitude(),latitude[counter], longitude[counter],
    results);
    distances[counter]=results[0];
    counter++;}

Appendix 3: FQL to get nearby friends academics history details to be used in clustering (see online version for colours)

for ( int i = 0, size = distances.length; i < size && distances[i]!=0 ; i++ ){
    if( distances[i]<float(1)1500)
    {
        neededIndex[counter3]=i;
        Nearby_friend_id[counter3]= author_uid_array[i] ;
        counter3++;
    }
} for ( int i = 0, size = counter3; i < size ; i++ ){
    if(i!=size-1)
    {
query+= "uid="+Nearby_friend_id[i]+" or ";
} 
else 
{
query+= "uid="+Nearby_friend_id[i];
}
} 
Bundle params2 = new Bundle();
params2.putString("method", "fql.query");
params2.putString("query", "SELECT uid,name,education FROM user WHERE "+query");
String response2 = utility.mFacebook.request(params2);
response2 = 
{"data":" + response2 + "
"};
JSONObject json2 = Util.parseJson(response2);
data2 = json2.getJSONArray("data");
for ( int i = 0, size2 = data2.length(); i <size2; i++)
{
JSONObject friend2 = data2.getObject(i);
Nearby_friend_Name[i]= friend2.getString("name");
}

Appendix 4: Code for listing of nearby friends with their location, when user checks-in (see online version for colours)

Button mGetNearbyFriends = (Button) findViewById(R.id.get_nearby_friends);
mGetNearbyFriends.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        try {
            TextView friends_Locations = (TextView) findViewById(R.id.friends_Locations);
            friends_Locations.setText("/");
            String jsonUser = null;
            for ( int i = 0, size = counter3; i < size; i++)
            {
                jsonUser = Utility.mFacebook.request(""+checkin_id[neededIndex[i]]); 
                obj = Util.parseJson(jsonUser);
                placeName[i]= obj.optString("place").getString("name");
                created time[i]= obj.getString("created_time");
                friends_Locations.append(Nearby_friend_Name[i] +" checked in "+placeName[i] +" at "+
                created time[i]+" "");
            }
            catch (MalformedURLException e) {} 
            catch (IOException e) {} 
            catch (FacebookError e) {} 
            catch (JSONException e) {} 
        } 
        catch (MalformedURLException e) {} 
        catch (IOException e) {} 
        catch (FacebookError e) {} 
        catch (JSONException e) {} 
    }
});
Appendix 5: Code to calculate rate of calories burned per pound of body weight
(see online version for colours)

```java
private void getDistanceAndCalories() {
    TextView DistanceMoved = (TextView) findViewById(R.id.distance_moved);
    TextView ActiveTime = (TextView) findViewById(R.id.active_time);
    TextView WalkingBurnedCalories = (TextView) findViewById(R.id.burned_calories);
    EditText Weight = (EditText) findViewById(R.id.weight);
    DistanceMoved.setText("Total Distance you walked : " + TotalDistance);
    activeTime = (counter-1)*30/60; //minutes
    ActiveTime.setText("Active time : " + activeTime + " minutes or ");
    totalDistanceInMiles=TotalDistance*(float)1.609344;
    A=(float) 0.0195;
    B=(float) 0.004376;
    C=(float) 0.00245;
    K = 1;
    weightInKgs=(float)Double.parseDouble(Weight.getText().toString());
    weightInPounds=weightInKgs*(float)2.20462;
    D=(float)(Math.pow(weightInPounds/145, 0.425)*0.000801)*weightInPounds;
    V=(totalDistanceInMiles/activeTimeInHours); //Walking Speed (mph) - Limited to a minimum of
    1 mph and a maximum of 5 mph
    ratePerPound= (float)(A+B*V)+(C*Math.pow(V,2)+(K*D)*Math.pow(V,3));
    walkingBurnedCalories= ratePerPound*weightInPounds;
    WalkingBurnedCalories.setText("Walking burned calories : "+ walkingBurnedCalories + " calories in ");
}
```

Appendix 6: Code for user friends (see online version for colours)

```java
String query = "select name, current_location, uid, pic_square from user where uid in (select uid2 from friend where uid1=me()) order by name";
Bundle params = new Bundle();
params.putString("method", "fol.query");
params.putString("query", query);Utility.mAsyncRunner.request(null, params, new TrustedFriendsRequestListener());
```

Appendix 7: Code to get user trusted friends list (see online version for colours)

```java
string query = "select name, current_location, uid, pic_square from user where uid in (SELECT uid FROM friendlist_member WHERE friendlist_member WHERE uid="+ trustedFriendListId + ") order by name";
Bundle params = new Bundle();
params.putString("method", "fol.query");
params.putString("query", query);
Utility.mAsyncRunner.request(null, params, new assignCloacTrustedRequestListener());
```
Appendix 8: Code for sharing weather temperature - enforced with trusted and semi-trusted friends policy (see online version for colours)

```java
private void share() {
    String s="";
    Bundle params = new Bundle();
    params.putString("message", temperature);
    JSONObject privacy = new JSONObject();
    try {
        privacy.put("value", "CUSTOM");
        privacy.put("friends", "SOME_FRIENDS");
        privacy.put("allow", trustedList="","+semiTrustedList(d);
        privacy.put("deny", untrustedList(d);
    } catch (JSONException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    params.putString("privacy", privacy.toString());
    try {
        Utility.selectAll(RESULT, params, "POST");
        Toast toast;
        toast = Toast.makeText(this,"Your status has been updated", Toast.LENGTH_LONG);
        toast.show();
    } catch (FileNotFoundException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (MalformedURLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```