z^3
Storage and display of sleep-related data

Pierre Azalbert
EE4-67 Mobile Healthcare and Machine Learning
Department of Electrical and Electronic Engineering, Imperial College London, SW7 2AZ
Supervisor: Dr. Yiannis Demiris

***Abstract*—z^3 is a sleep monitoring system that records several types of information about the user and their sleeping environment, in order to provide tailored advice on how to improve their sleep. The system is based on a mobile application that connects to a TI SensorTag 2.0 to collect temperature, humidity, light and noise level data. It also gathers information about the user's lifestyle, sleep environment and moods through surveys implemented in the mobile application. Due to the large amount of data collected every day by the mobile application it is important to find the best way to store this data, and to display it to the user in a useful and engaging way. We have implemented a database local to the mobile device using Realm for iOS, and we will use the Realm Object Server to synchronise local databases on various devices with a centralised remote database. Once that is done we will build graphs to display user information on the mobile application.**

I. Introduction

The TI SensorTag 2.0 [1] does not have any embedded memory, it can only stream live data, therefore we need to store this information locally on the device running the mobile application. All the information collected by the mobile application and the connected sensor is fed to the recommender algorithm which will generate recommendations and weightings associated to the data collected. Consequently we need to make sure that the data is well organised, clearly labelled and easily accessible by other components of the system, so that all its value can be exploited in order to satisfy the user's needs.

In addition to that, the mobile application will be able to send some user-related statistics to a remote server in order to generate statistics about all the users in the z^3 network. This information will be pushed back to each user's device, and they must be able to quickly and easily review this data along with their personal results.

This report will focus on the design and implementation of local and remote databases to store user- and network-related information, as well as ways of displaying this information. We will also look at the integration of user surveys in the mobile application.

II. Existing solutions

There are various commercial applications of sleep monitoring solutions already available to users. Some of them are implemented simply as mobile application, using the hardware embedded in the mobile device running the application. Some others are designed as a system where the mobile application connects to external devices to monitor sleep-related information. We will review a few examples of those two types of solutions, in order to understand what kind of data they collect, and what kind of information is communicated to the user.

A. Mobile applications

Amongst the various mobile applications available on App Stores, some of the most famous and successful are Sleep Cycle App [2], Pillow App [3] and My Sleep Bot [4]. Pillow App and My Sleep Bot track sleep through movement monitoring, using the accelerometer inside the user's mobile device which therefore needs to be placed on the bed. However, Sleep Cycle now also has the ability to do the same thing using the device's microphone, if it is placed on a table next to the bed and facing the user. This is a technology similar to that mentioned in [5], which suggests that using the microphone inside the TI SensorTag we could potentially collect objective data about sleep quality. Additionally, My Sleep Bot and Pillow App can use the microphone to record specific sound events during the night, which the user can play back, to see if they are sounds which might have affected their sleep.

All of the three applications allow the user to record custom text notes that are attached to each day, to help him/her remember any specific factors that might have affected their sleep. Sleep Cycle also allows the user to record their mood when they wake up. We are also planning to implement this kind of feature in our system because it will help the recommender algorithm in generating recommendations.

Sleep Cycle and Pillow App provide detailed information about sleep patterns, detailing each sleep phase (deep, light, REM) and its duration. They provide graphs that cover days, weeks or months, showing quantities such as average sleep quality, average time time bed, average time to bed and average wake up time. Sleep Cycle also shows user notes linked to decreased or increased sleep quality, and sleep quality per day of week.

B. Connected devices

The higher end solutions offered for sleep monitoring are more expensive than the self-sufficient mobile applications, and rely on external hardware to monitor sleep-related information. Beddit [6] is a system that uses a bed sensor placed under the mattress to record sleep phases, heart rate, breaths per minute and snoring, as well as temperature and humidity. The companion mobile application can automatically start or stop data logging and it allows the user to record personal notes (tags) to label their sleep. It is possible to export the data to medical databases for reviewing by a doctor and in exchange of a fee the user can receive a detailed sleep report covering 2 weeks of data.

Another product in this range is the Withings Aura [7], which is like an enhanced alarm clock. It contains a light that changes colour at night to encourage the production of sleep hormones, and that can simulate sunrise in the morning. The embedded speakers are connected to Spotify [8] to play either sleep-inducing sounds in the evening or energizing music in the morning. The Aura can also continuously monitor temperature, luminosity and sound levels. The user can also buy an accessory that is placed under the bed to monitor sleep cycles and sleep duration.

III. Design requirements & resources

A. General requirements

Based on our system design and the features we observed in existing solutions, we want to find the best way to store all the data collected by our system in a simple and efficient way. It must be easily accessible by other components of the system such as the recommender algorithm or the plots that display the user data. We also want to be able to link groups of data together, for example link one night's sensor samples with the user tags selected on the corresponding day.

In addition to storing the data locally on the user's mobile device we want to be able to upload it on a remote server in order to analyse trends in groups of users which can then be communicated to individual users.

Finally once we have designed and implemented all the databases (local and remote) for our system's data, we will have to build a graphical user interface that contains a variety of plots, similar to those found in existing solutions, in order to communicate useful information to the user.

B. Resources

Since our system will be developed for Apple devices running on iOS, we had to find ways of implementing local data storage using the Swift programming language. According to [9] there are various options including SQLite [10], Core Data and Realm [11]. Core Data is native to iOS, it is fast but takes up a lot of memory, while SQLite consumes less space to the cost of less speed. The main difference is that Core Data is an object oriented database whereas SQLite stores data in tables. Realm is also object oriented, but compared to Core Data it consumes little memory, is faster than SQLite, and its syntax is very simple. We will use Realm to implement our local database because it will allow us to develop it very quickly and because its object oriented architecture is a nice way of linking groups of data together.

Another interesting feature of Realm is the Realm Object Server [12] which can be setup on any computer to receive and send data to/from mobile devices. Any changes on either the computer or the mobile device is quickly propagated on all devices. We think this would be a good solution to build our remote database.

IV. Implementation

A. Design Overview

Both the local and remote data storage will be implemented using Realm, therefore we will define a database model based on several objects which correspond to different groups of data such as users, sensor data, etc. There will be local functions to process the locally stored data and turn it into graphs, and there will be functions running on the remote server to process network data and send it back to individual users. The graphs will display historical data over days/weeks/months as well as data for a specific night. Processing the data will allow us to show long term trends in sleep quality with details on the effects of every user-defined tag.

B. Local data storage

Instead of starting completely from scratch we used a demo iOS application found online [13] which already implements the BLE (Bluetooth Low Energy) communication between an iPhone and the TI SensorTag 2. It is a very simple one-screen application which reads temperature and humidity from the sensor and shows the values on the iPhone screen. Using tutorials and demo iOS apps provided on the Realm website [14] we inserted Realm code inside the BLE demo application to store the temperature and humidity samples over time. We defined a real model with various objects, which are linked to each other as shown in figure 1.

The User object (figure 2) contains a unique key and a name, and a pointer to a list of sensorDataObjects. This list is appended every time new samples are read from the sensor. Each sensorDataObject (figure 2) contains the sensor ID as well as all the samples read from the sensor. Right now we only store temperature and humidity but the application will be extended to other quantities for the final version of the system. We will also have to define other objects such as surveyDataObject and recomDataObject to store survey answers and recommender results respectively. The links between objects and lists of objects are very convenient when querying the database. Figure 3 shows what is returned when we print a single sensorDataObject in the console.

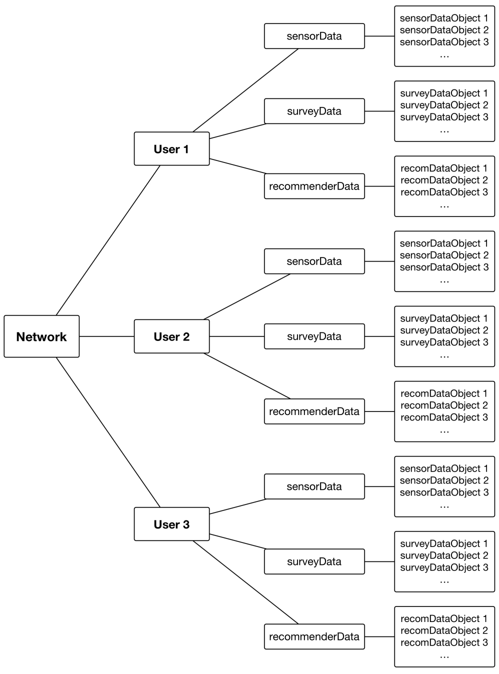
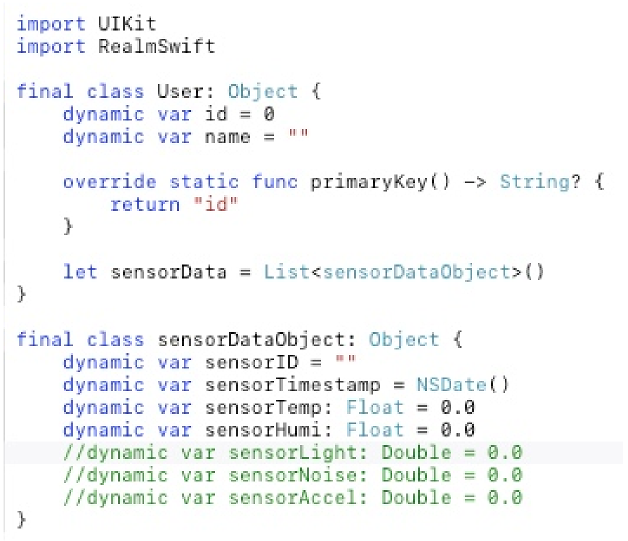
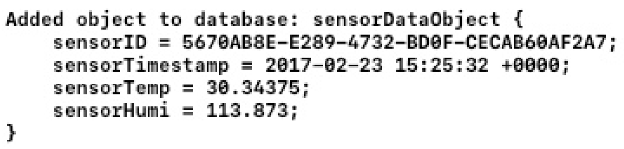


Fig. 1. Database Object-Oriented Model



```swift
import UIKit
import RealmSwift

final class User: Object {
    dynamic var id = 0
    dynamic var name = ""

    override static func primaryKey() -> String? {
        return "id"
    }

    let sensorData = List<sensorDataObject>()
}

final class sensorDataObject: Object {
    dynamic var sensorID = ""
    dynamic var sensorTimestamp = NSDate()
    dynamic var sensorTemp: Float = 0.0
    dynamic var sensorHumi: Float = 0.0
    //dynamic var sensorLight: Double = 0.0
    //dynamic var sensorNoise: Double = 0.0
    //dynamic var sensorAccel: Double = 0.0
}
```

Fig. 2. Realm Object-Oriented Model Implementation

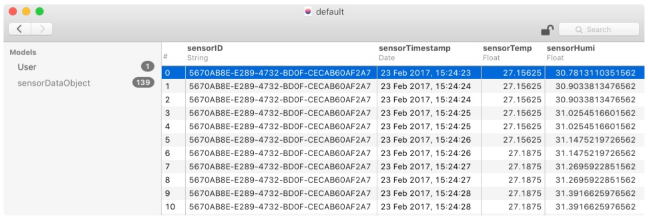


```
Added object to database: sensorDataObject {
    sensorID = 5670AB8E-E289-4732-BD0F-CECAB60AF2A7;
    sensorTimestamp = 2017-02-23 15:25:32 +0000;
    sensorTemp = 30.34375;
    sensorHumi = 113.873;
}
```

Fig. 3. Console output of an instance of sensorDataObject

C. *Network data storage*

Using the Realm Object Server, we can tell our iOS app to synchronise with a remote machine. We have setup an AWS (Amazon Web Services) [15] instance running Ubuntu, and installed the Realm Object Server on it. So far we have been able to send some test data using the demo applications provided by the Realm documentation, we will implement the remote connection in the BLE application as soon as possible. Once the data transfer has been setup we will be able to write a program running on the AWS machine to process all the data received and store network statistics in a newtorkDataObject that will be sent back to individual users for local storage. Figure 4 is an example of what the data looks like when stored on a Realm Object Server (which is the same as the Realm file stored on the iPhone's memory).



#	sensorID String	sensorTimestamp Date	sensorTemp Float	sensorHumi Float
0	5670AB8E-E289-4732-BD0F-CECAB60AF2A7	23 Feb 2017, 15:24:23	27.15625	30.7813110351562
1	5670AB8E-E289-4732-BD0F-CECAB60AF2A7	23 Feb 2017, 15:24:24	27.15625	30.9033813476562
2	5670AB8E-E289-4732-BD0F-CECAB60AF2A7	23 Feb 2017, 15:24:24	27.15625	30.9033813476562
3	5670AB8E-E289-4732-BD0F-CECAB60AF2A7	23 Feb 2017, 15:24:25	27.15625	31.0254516601562
4	5670AB8E-E289-4732-BD0F-CECAB60AF2A7	23 Feb 2017, 15:24:25	27.15625	31.0254516601562
5	5670AB8E-E289-4732-BD0F-CECAB60AF2A7	23 Feb 2017, 15:24:26	27.15625	31.1475219726562
6	5670AB8E-E289-4732-BD0F-CECAB60AF2A7	23 Feb 2017, 15:24:26	27.1875	31.1475219726562
7	5670AB8E-E289-4732-BD0F-CECAB60AF2A7	23 Feb 2017, 15:24:27	27.1875	31.2695922851562
8	5670AB8E-E289-4732-BD0F-CECAB60AF2A7	23 Feb 2017, 15:24:27	27.1875	31.2695922851562
9	5670AB8E-E289-4732-BD0F-CECAB60AF2A7	23 Feb 2017, 15:24:28	27.1875	31.3916625976562
10	5670AB8E-E289-4732-BD0F-CECAB60AF2A7	23 Feb 2017, 15:24:28	27.1875	31.3916625976562

Fig. 4. Contents of a Realm Object Server

D. *Future work*

To reach the final design of our system we still need to work on the local database model, as some of the Realm objects haven't been implemented yet. We will also need develop the user interface for the surveys that the user needs to fill in before going to bed and after waking up. We will probably use the Apple ResearchKit framework [16] which is open source, in order to quickly implement the survey functionality.

As soon as the remote database is up and running, we will focus on writing code to display graphs in the mobile application, to display things such as average sleep time, average wake up time, tags with negative impact on sleep, etc. There are a variety of libraries [17] and tutorials [18] [19] readily available to build graphs on iOS, it will mostly be a matter of deciding exactly what kind of information we want to show the user.

V. Conclusion

In this report we have reviewed existing commercial solutions for sleep monitoring, in order to understand what kind of data they collect and what information they give to the user. We have started developing our own iOS application based on existing tutorials; it can already retrieve data samples from the TI SensorTag and store all of this data in an object-oriented model based on the Realm framework. We will finish implementing the local database as soon as possible so that other modules such as the recommender algorithm can be integrated to the mobile application. After that, we will work on the remote server and the data visualisation.

REFERENCES

[1] T. Instruments, "Simplelink sensortag." http://www.ti.com/ww/en/wireless_connectivity/sensortag2015/index.html. Online; Accessed: 2017-02-02.

[2] N. AB, "Sleep cycle alarm clock." https://www.sleepcycle.com. Online; Accessed: 2017-02-24.

[3] N. Interactive, "Pillow: Sleep tracking alarm clock with powernaps and audio recordings." https://neybox.com/pillow/, 2011. Online; Accessed: 2017-02-24.

[4] SleepBot, "Sleepbot." https://www.mysleepbot.com. Online; Accessed: 2017-02-24.

[5] T. Hao, G. Xing, and G. Zhou, "Isleep: Unobtrusive sleep quality monitoring using smartphones." http://www.cse.buffalo.edu/~lusu/cse721/papers/iSleep%20Unobtrusive%20Sleep%20Quality%20Monitoring%20using%20Smartphones.pdf, 2013. Online; Accessed: 2017-01-30.

[6] Beddit, "Beddit sleep tracker." http://www.beddit.com, 2016. Online; Accessed: 2017-01-30.

[7] W. SA, "Withings aura." http://www.withings.com/us/en/products/aura?, 2009. Online; Accessed: 2017-02-24.

[8] Spotify, "Spotify." https://www.spotify.com/uk/. Online; Accessed: 2017-02-24.

[9] O. Prusak, "Ios databases: Sqllite vs. core data vs. realm." https://rollout.io/blog/ios-databases-sqllite-core-data-realm/, 02 2016. Online; Accessed: 2017-02-24.

[10] stephencelis, "Stephencelis/sqlite.swift: A type-safe, swift-language layer over sqlite3." https://github.com/stephencelis/SQLite.swift, 01 2017. Online; Accessed: 2017-02-24.

[11] Realm, "Realm documentation for swift." https://realm.io/docs/swift/latest/\#getting-started. Online; Accessed: 2017-02-24.

[12] Realm, "Realm object server." https://realm.io/docs/realm-object-server, 2014. Online; Accessed: 2017-02-24.

[13] E. K. Stone, "Zero to ble on ios part two - swift edition." https://www.cloudcity.io/blog/2016/09/09/zero-to-ble-on-ios--part-two---swift-edition/, 09 2016. Online; Accessed: 2017-02-24.

[14] Realm, "Building your first realm mobile platform ios app." https://realm.io/docs/tutorials/realmtasks/, 2014. Online; Accessed: 2017-02-24.

[15] Amazon, "Amazon web services." https://aws.amazon.com. Online; Accessed: 2017-02-24.

[16] A. Inc, "Researchkit." http://researchkit.org. Online; Accessed: 2017-02-24.

[17] ameizi, "Ameizi/awesome-ios-chart: A curated list of awesome ios chart libraries, including objective-c and swift," 07 2016.

[18] S. Korpela, "Using realm and charts with swift 3 in ios 10." https://medium.com/@skoli/using-realm-and-charts-with-swift-3-in-ios-10-40c42e3838c0\#.hrv612om1, 10 2016. Online; Accessed: 2017-02-24.

[19] J. Echessa, "How to use ios charts api to create beautiful charts in swift." https://www.appcoda.com/ios-charts-api-tutorial/, 06 2015. Online; Accessed: 2017-02-24.