UBDROID: A TOOL FOR MONITORING SMARTPHONE APPLICATION USAGE FOR USER BEHAVIOR ANALYSIS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS INSTITUTE
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ERKAM AKKURT

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF INFORMATION SYSTEMS

JULY 2015

UBDROID: A TOOL FOR MONITORING SMARTPHONE APPLICATION USAGE FOR USER BEHAVIOR ANALYSIS

Submitted by **Erkam AKKURT** in partial fulfillment of the requirements for the degree of **Master of Science in Information Systems**, **Middle East Technical University** by,

Prof. Dr.Nazife Baykal _____
Director, **Informatics Institute**

Prof. Dr.Yasemin Yardımcı Çetin _____
Head of Department, **Information Systems**

Assoc. Prof. Dr. Alptekin Temizel _____
Supervisor, **Modeling and Simulation**

Assist. Prof. Dr. Tuğba Taşkaya Temizel _____
Co-supervisor, **Information Systems**

Examining Committee Members:

Assoc. Prof. Dr. Altan Koçyiğit _____
Information Systems, METU

Assoc. Prof. Dr. Alptekin Temizel _____
Modeling and Simulation, METU

Dr. Haluk Altunel _____
SOFTTECH

Assoc. Prof. Dr. Aysu Betin Can _____
Information Systems, METU

Assist. Prof. Dr. Erhan Eren _____
Information Systems, METU

Date: **06/07/2015**

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and result that are not original to this work.

Name, Last Name : Erkam Akkurt

Signature :

ABSTRACT

UBDROID: A TOOL FOR MONITORING SMARTPHONE APPLICATION USAGE FOR USER BEHAVIOR ANALYSIS

Akkurt, Erkam

M.S., Department of Information Systems

Supervisor: Assoc. Prof. Dr. Alptekin Temizel

Co-Supervisor: Assist. Prof. Dr. Tuğba Taşkaya Temizel

January 2015, 63 pages

UBDroid is a multilayer tool for monitoring application usage on Android platform. UBDroid consists of an Android client application and a server application. The client application keeps track of user interactions by collecting start and running time of applications and sensor data. The data are collected on the device and subsequently sent to a remote server on a schedule and as a result the system does not require constant network connection. The server application gathers application information such as category and rating from Google Play and processes the collected data. UBDroid provides an energy efficient system for collecting user data which is valuable for user behavior analysis.

Keywords: User behavior analysis, mobile application analysis, Android application usage, mobile sensing.

ÖZ

UBDROID: KULLANICI DAVRANIŞ ANALİZİ İÇİN AKILLI TELEFON UYGULAMALARI KULLANIM İZLEME ARACI

Akkurt, Erkam

Yüksek Lisans, Bilişim Sistemleri Bölümü

Tez Yöneticisi: Doç. Dr. Alptekin Temizel
Eş Danışman: Yrd. Doç. Dr. Tuğba Taşkaya Temizel

Ocak 2015, 63 sayfa

UBDroid, Android platformunda uygulama kullanımını izlemek için çok katmanlı bir araçtır. UBDroid Android istemci uygulaması ve sunucu uygulamasından oluşur. İstemci uygulaması, uygulamaların başlatılma zamanı ve çalışma süresi ile algılayıcı verilerini kaydeder. Aygıt üzerinde toplanan veri daha sonra uzak sunucuya belirli aralıklarla aktarıldığından sistem sürekli ağ bağlantısı gerektirmemektedir. Sunucu uygulaması Google Play'den uygulama puan ve kategori gibi bilgileri edinir ve toplanan veriyi işler. UBDroid kullanıcı davranışı analizi için kullanım bilgisi toplayan enerji verimli bir sistem sağlamaktadır.

Anahtar Sözcükler: Kullanıcı davranış analizi, mobil uygulama analizi, Android uygulama kullanımı, mobil algılama.

To my family

ACKNOWLEDGMENTS

I want to express my gratitude to my supervisor Assoc. Prof. Dr. Alptekin Temizel for his encouraging, advice and guidance in this thesis study.

I am very thankful to Assist. Prof. Dr. Tuğba Taşkaya Temizel for her important insights that addresses requirements of our study.

I also would like to thank everyone who keeps my motivation up with their support and encouragement during the study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

SDK	Software Development Kit
GPS	Global Positioning System
LBS	Location-based service
SMS	Short messaging system
3G	3rd Generation
IMEI	International Mobile Equipment Identity
DDMS	Dalvik Debug Monitor Server
JSON	JavaScript Object Notation
ADT	Android Development Toolkit
UTC	Coordinated Universal Time
REST	Representational State Transfer
DTO	Data Transfer Object
ORM	Object Relational Mapping
CRUD	Create, Read, Update, Delete

CHAPTER 1

INTRODUCTION

1.1 Motivation

Smartphone usage has increased significantly during the last decade. In 2007, Apple's iPhone had the greatest impact on the smartphone market by its innovative features. It attracted attention with a very responsive user interface powered by capacitive touch screen. It provided multimedia and synchronization features without bothering users with the particulars of a file system unlike Windows Mobile and Symbian. In 2008, mobile application distribution platform, App Store, has been added to iPhone ecosystem. It was a revolutionary feature that was followed by its competitors; Google Play (Android Market) in 2008, Nokia Store in 2009, Blackberry World in 2009, Windows Phone Store in 2010 [1]. Mobile application distribution platforms provide control over application developers. They helped detecting and preventing distribution of malicious software. They, also, created a great market for developers. Apple reported that total number of application downloads over App Store hit 50 billion in May 16, 2013 [2] and in only 2013, customers spent over $10 billion on App Store [3]. Even though Apple's iPhone initiated the revolution, Android powered smartphones are dominating the market as reported in [4] [5] [6]. In 2014, 80% of sales were Android smartphones while Apple amounted to 15% of sales. Android usage is far beyond its competitors by the help of wide variety of smartphones on the market.

Availability of high speed internet on smartphones has changed user habits. Users started to spend more time on smartphones than computers for browsing. Popular social networking, e-commerce, news, financing web sites have mobile applications.

Mobile applications provide better user interface and better usage of smartphone resources. Social networking applications can directly access camera and allow sharing photo in seconds while web browser bothers with file upload interface for same functionality. Banking applications started to authenticate users by finger print sensor. On the other hand, their web versions require two level security. Mobile applications are also able to notify user even if device is in sleep mode. Mobile application is preferred rather than web page while users are moving toward mobile devices. By the effects of changing user habits, on-site web analytics adopted to mobile applications. Web-based user analysis methods are limited to data gathered from session, cookies, web surveys; but mobile user analysis methods includes on-device metrics such as location, running tasks; call logs, contacts, SMS, physical sensor data. Studies show that these on-device metrics can be used to create valuable data in mobile sensing applications [7] [8] [9] [10], mobile user analysis [11] [12] [13] [14] [15] and mobile application analysis [16] [17] [18]. The studies share a common approach; domain specific data collection, centralization of the collected data and data processing or mining.

1.2 Scope

The purpose of the study is to introduce a tool to create valuable database for mobile user behavior analysis. The main focus of the tool, UBDroid, is collection of application usage information from Android smartphone users. Moreover, UBDroid allows reading sensor values for a period time. Additionally, UBDroid allows user participation by messaging and online surveys. UBDroid ought to centralize collected data and make further process on server such as gathering application information from Google Play. In addition, UBDroid consumes minimal resources on smartphone in order to avoid affecting the overall performance and increasing battery consumption of the device.

UBDroid is intended to gather usage data from a large number of participants. It is not designed to run on a special hardware or modified versions of Android, so data collection does not require an experimental environment. Participants can install UBDroid to their smartphones and continue their daily usage habits; it runs in the

background whenever smartphone runs. UBDroid has no need for super user access or modified Android sources, so it is affected by operating system limits; user input events cannot be logged.

UBDroid does not only collect data from user, but also fetches further application information from Google Play including category, rating, rating count and number of downloads. UBDroid does not require user participation, but gives opportunity to perform online surveys. Application usage data, supported by Google Play statistics and online user surveys, creates value for further data mining operations.

1.3 Outline

This thesis organized in five chapters as follows:

- **Chapter 2** focuses on the concept of mobile sensing, and elaborates mobile sensing frameworks. Additionally, mobile user behavior analysis and mobile application analysis studies are examined. Lastly, common android battery drain problems are presented.
- **Chapter 3** introduces detailed design of proposed system. Also, major system objectives, which affect system design, are listed.
- **Chapter 4** introduces resulting application usage data.
- **Chapter 5** includes conclusion which implies evaluation of UBDroid and possible future extensions.

CHAPTER 2

LITERATURE REVIEW

Many analysis systems rely on data collection and data mining techniques that is specialized to their context. UBDroid has similar data collection methods with mobile sensing approach, but it is not only focused on physical sensor data. UBDroid aims to collect valuable data for user behavior analysis based on application usage information. UBDroid is energy efficient, so it does not cause significant battery drain. By considering these features of UBDroid, related works are categorized and examined in the four subsections: mobile sensing, user behavior analysis, application analysis, android battery management.

2.1 Mobile Sensing

A modern smartphone is more powerful than a super-computer of a few decades ago [19]. By the help of its computing power, smartphones are taking place of personal computers. Users can perform tasks of web browsing, mailing, social networking, shopping, gaming, etc. on their smartphone instead of personal computer.

Even though smartphones' computing power dramatically increasing, it is not the only reason to why smartphones are intensively used. Mobility and connectivity has important role on the popularity of smartphones. In early 2014, it was reported that mobile internet usage exceeded internet usage from PCs [20]. Smartphones could connect to the Web via Wi-Fi or Bluetooth since early 2000s, but 3G had the greatest impact on increasing connectivity and data rates.

Modern smartphones are equipped with several sensors as listed in Table 1. These sensors and mobile operating systems that provide API for 3rd party application developers enable applications to benefit from them. Smartphones, with increased connectivity capabilities, enriched by the sensors, became suitable for the concept of

mobile sensing. Mobile sensing can be described as fetching data from mobile devices by the help of their sensors. Smartphones are becoming increasingly popular devices for mobile sensing frameworks. However, there exist other applications such as wearables, smartwatches, and heart rate trackers.

Table 1 - List of sensors on modern smartphones [21]

Sensor	**Function**
Accelerometer	Measures the acceleration force in m/s2 that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.
Barometer/Pressure	Measures the ambient air pressure in hPa or mbar.
GPS	Measures the position on the earth.
Gravity	Measures the force of gravity in m/s2 that is applied to a device on all three physical axes (x, y, z).
Gyroscope	Measures a device's rate of rotation in rad/s around each of the three physical axes (x, y, and z).
Light	Measures the ambient light level (illumination) in lx.
Linear acceleration	Measures the acceleration force in m/s2 that is applied to a device on all three physical axes (x, y, and z), excluding the force of gravity.
Magnetometer	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in μT.
Microphone	Measures sound.
Orientation	Measures degrees of rotation that a device makes around all three physical axes (x, y, z).
Proximity	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.
Relative humidity	Measures the relative ambient humidity in percent (%).
Temperature	Measures the temperature of the device in degrees Celsius (°C).

Lane et al. divides sensing systems into two types; opportunistic or participatory [22]. Opportunistic sensing requires automated data collection on the device which requires little or no user involvement. On the other hand, participatory sensing relies on participation of users. All knowledge belongs to users at critical steps of data collection. UBDroid adopts opportunistic sensing approach by collecting application usage and sensor data, but it also allows participatory sensing by online surveys.

Lane et al. expresses that mobile sensing can be applied for various area at different scales such as social networking, environmental monitoring, health and well-being, transportation, application stores. [9] Lu et al. proposes continuous sensing engine, called Jigsaw, which is able to log caloric expenditure, daily activities, significant places and transportation methods of user [10]. Jigsaw relies on data collected from accelerometer, microphone and GPS sensors. It is able to run on Nokia N95 and jailbroken iPhone. It runs entirely on mobile device which and does not use external server. Even though mobile sensing systems apply for specific domains, the two mobile sensing frameworks are examined: SynchoSmart Framework, Funf Open Sensing Framework.

Kepucka proposes a framework, SynchoSmart, for emotion analysis of audience during particular events such as movies, sport tournaments, and cultural shows [7]. SynchoSmart relies on client-server model whereas clients are Android smartphones. Key features of SynchoSmart are:

- Accelerometer, linear accelerometer, gyroscope, magnetometer, orientation, rotation vector and microphone are used for data collection.
- It uses collect then transmit approach. Collected data is stored in local storage of smartphone, and then transmitted to the server.
- It centralizes data collection and processing in order to provide data integrity and minimize workload on smartphone.
- It synchronizes data collection. Data is collected from multiple sources whose time information may differ. Synchronization is performed when the data is centralized.

SynchoSmart gather data for emotion analysis, but its data synchronization and centralization methods can be applied to other domains.

Funf is an open-source mobile sensing framework that can collect data from all available sensors on smartphone [8]. It is able to run on Android smartphone. It can also be added to projects as library, so it reduces application development effort. Key features of Funf are listed below.

- Funf reports call log, SMS log, browser history, running apps, installed apps, battery status in addition to the data gathered from all available sensors.
- It keeps collected data in local storage of smartphone, and then transmits to server when internet connection is available.
- It encrypts locally stored data.
- It allows automatic or manual data upload.

Funf has advanced data collection capabilities, but it does not offer built-in data centralization method. Funf is highly configurable, but enabling all data collection methods causes battery drain.

2.2 Mobile User Behavior Analysis

User analysis can be defined as identification of user characteristics in order to improve quality of service. The identification process is typically done by a set of measurement that should be performed on target users. Verkasalo presents MobiTrack Framework for mobile user measurements which focus on data collection [11]. The study implies the importance of on-device measurements. MobiTrack has multiplatform support; it can run on Symbian, Windows Mobile, Android and Blackberry devices. MobiTrack has three sources of data; behavioral measurements, contextual surveys and web-based surveys. Behavioral measurements consist of device-based metrics. These metrics are gathered from call usage, SMS usage, application usage, browsing usage, data service usage. Contextual surveys are on-device questionnaires about user feedback or user satisfaction. Web-based surveys used to determine user needs. As Verkalaso stated, one of the key advantages of

MobiTrack is its capability to collect a comprehensive set of data from real environments of users.

Chen et al. proposes a log collection service for analyzing mobile user behavior [12]. Proposed service can run on Android smartphones. The service keeps all user operations which are extracted from system logs by using *logcat* tool of Android SDK. In addition, location information is attached to log data. Since Global Positioning System uses a lot of power, Location Based Service is used to determine location even if it does not provide location information as sensitive as GPS. The collected user log data is transmitted to the server on a schedule so that the service does not require internet connection all the time. There is also log query service to allow fetching collected data filter by user, activity or time. Chen et al. states that the proposed service platform can be used to gather log for specific type of activity, but there is no categorization mechanism for activities. In other words, who query the data should know the social networking applications in order to fetch logs of social networking activities.

LiKamWai et al. introduces smartphone software, MoodScope, which predicts current mood of user based on smartphone usage [13]. MoodScope runs as background service and monitors user activities without user interference. It infers mood of the user by interpreting SMS, email, phone call, application usage, web browsing history and location data. MoodScope can run on jailbroken iPhone in order to overcome operating system limitations.

Pejovic et al. developed a model, called InterruptMe, for analyzing interruptibility of smartphone users [14]. The model seeks to identify suitable moments of users in order to increase the impact of delivered information. The accessibility of users is affected by location, time of day, their activity and emotions. The model has three objectives; reaction presence, timely reaction and sentiment. In other words, users should react to the delivered information; they should react in a reasonable time interval; they should be willing to react. The model benefits from Bluetooth and Wi-Fi environments, GPS coordinates and accelerometer data as well as previous user

reactions. The proposed model helps increasing user involvement in participatory mobile sensing systems.

Yan et al. proposes a personalized application recommendation system, AppJoy [15]. The system recommends applications to smartphone users that they might like. The traditional way relies on application download history and ratings of user. Application download history does not imply that users like the application, they might be only trying the application out. User ratings provide better result, but most users are not willing to rate. AppJoy calculates usage score from application usage information by proposed recency, frequency, and duration model. Similar applications are matched by usage scores of the other users who used the application. AppJoy runs without user interaction by analyzing application usage data. It relies on client-server architecture where client application runs on Android platform as a background service. AppJoy client periodically uploads application usage data to the server. At server side, application recommendations are created according to statistics generated from application usage data.

Mobile user analysis depends on data collection on mobile devices and processing the data. The types of the data to be collected differ according to research context. Mobile sensing and user analysis studies are summarized in Table 2 – Summary of mobile sensing and user analysis studies. UBDroid focuses on application usage collection and categorization of applications by crawling Google Play web page. In addition, it is able to collect sensor data. As key features, UBDroid allows participatory data collection and provides administrative web client.

Table 2 – Summary of mobile sensing and user analysis studies

Study	Intent	Data Collection				Key Features	Platform
		App. Usage	Sensor	Other	Participatory		
Jigsaw	Monitoring human activity and context	-	GPS, Accelerometer, Microphone	-	No	•Does not use server •Capable of logging daily activities and significant places	iOS (Jailbroken), Nokia N95
SynchoSmart	Emotion analysis of audience	-	Accelerometer, Gyroscope, Magnetometer,	-	No	•Client-server architecture •Most of the sensors	Android

	during particular events		Orientation, Rotation, Microphone			on modern smartphone can be used	
Funf	Developing mobile sensing capable applications	Running and installed apps	GPS, Accelerometer	Contacts, Call logs, SMS logs, Browsing history, Network status, Battery status	No	•Client-server architecture •Highly configurable, reduces development cost. •Automatic or manual data upload •Encrypted data storage	Android
MobiTrack	Analysis of user needs and improving user satisfaction	Running and installed apps	-	Call logs, SMS logs, Browsing history, Multimedia usage, Network status	Optional (On device and web based surveys)	•Client-server architecture •Automatic data upload •Supports multiple platforms	Android, Blackberry, Symbian S60, Windows Mobile
MoodScope	Predicting users' daily mood	Running apps	GPS	Call logs, SMS logs, Email logs, Browsing history	No	•Client-server architecture •Automatic data upload	iOS (Jailbroken)
InterruptMe	Determining opportune moments of user	-	GPS, Accelerometer	Network status	Yes (On device surveys and notifications)	•Client-server architecture •Automatic data upload	Android
AppJoy	Personalized application recommendation	Running and installed apps	GPS	-	No	•Client-server architecture •Automatic data upload •Crawls application information from Google Play	Android
UBDroid	Creating database for mobile user behaviour	Running and installed apps	GPS, Accelerometer, Gravity, Gyroscope,	-	Optional (On device surveys and notifications	• Client-server architecture •Automatic or manual data upload	Android

analysis	Magnetic Field	)	•Crawls application information from Google Play •Administrative web client

2.3 Mobile Application Analysis

Both user analysis and application analysis approaches use similar data collection techniques since they are related to each other by human-computer interaction. User analysis mainly relies on monitoring user actions on a set of applications, while application analysis relies on monitoring application responses while a group of users interact. In both approaches, the characteristics of the collected data are similar.

Wei et al. presents a multi-layer system, ProfileDroid, for monitoring and profiling Android applications [16]. ProfileDroid examines application in four layers; static layer, user interaction, operating system and network. Hardware usage and permissions are extracted from the static layer. User generated events, user inputs are captured from the user interaction layer. Operation system layer is used to get system calls, interacting services, file system operations. Network traffic of application is handled in the network layer. ProfileDroid extracts valuable data related to the analyzed application. However, analysis at the four layers cannot be done by a 3rd party application because of Android permissions. ProfileDroid overcomes the problem by connecting the device to Android SDK in debug mode. ProfileDroid creates valuable application analysis data, but it is not possible to apply its approach on opportunistic mobile sensing.

Lee et al. proposes a user interaction-based profiling system to overcome the limitations of development-level application debugging [17]. The system does not need source codes of application to be analyzed; instead it performs on-device data collection by kernel and Android framework level. Process-level hardware usage information is extracted at the kernel level. At framework level *Activity*, which is a part of the Android framework which interacts between user and application, is monitored. In addition, user input events are logged. The system stores on-device metrics and transmits the collected data to the server. Since the system runs on

multiple clients, time synchronization is performed while data is merged. The proposed system runs on modified Android 4.3 since kernel level data collection and user input events logging cannot be done with default sources of Android.

Falaki et al. introduces usage monitoring tool, called SystemSens, for unexpected application behavior detection [18]. It relies on client-server system architecture. Client application is able to run on unmodified Android 2.2 and above versions. It keeps track of CPU, memory, battery, network usage. The collected data is sent to server. It is reported that the size of data is 2.5MB on average for each user per day.

2.4 Android Battery Management

Battery drain problem commonly occurs by unconscious application developments that causes bugs or resource misuse. Ma et al. proposes eDoctor tool in order to detect and suggest solution for abnormal battery drain issues on Android applications [23]. As stated, they studied 50 cases with the accuracy of 94%. There are two major problems that attract attention: application bugs and overusing or misusing resources. Prevention of smartphone to enter sleep mode, leaving mediaserver running, leaving GPS enabled after getting location information are examples of application bugs. Overusing or misusing resources are commonly caused by fetching sensor data more frequently than necessary. Especially GPS consumes much more power than other sensors, so it should be used sparingly.

CHAPTER 3

RESEARCH METHODOLOGY

The scope of our study is to design, develop and evaluate a data collection system for mobile user behavior analysis. UBDroid adopts both opportunistic and participatory data collection approaches. The system runs on Android smartphones and collects data and allows performing surveys on mobile web pages. UBDroid focuses on collection of application usage data and categorization of these applications in order to create application usage database for mobile user analysis.

We study methodology in six chapters; system objectives, system use cases, system architecture and implementation decisions, server application design, web client application design and client application design. In system objectives, the major system requirements are listed. In system use cases, main usage scenario for data collection is studied. In system architecture and implementation decisions, overall system structure with used frameworks and libraries are studied. Application designs are studied in modules. Server application has five modules with the responsibilities of; user management, messaging, survey management, data synchronization and batch processing. Client application has three modules; user module, data collection module and message module. Web client, which is used for administration of the system, has four modules; user module, survey module, message module and batch module.

3.1 System Objectives

The following requirements were considered during the system design:

- **Application usage collection**: UBDroid is able to continuously watch foreground application changes and report package name, starting time and ending time of the applications.

- **Sensor data collection**: UBDroid can read four sensors accelerometer, gravity, gyroscope, magnetic field. In addition, current location can be gathered by using Google's Fused Location API if location services are enabled on the client. Sensors are not read continuously because of energy efficiency. Instead, sensor data collection can be triggered at arbitrary times by push notifications.
- **Participatory data collection**: UBDroid has survey creation and survey data collection capability in order to collect participatory user data anonymously.
- **Data integrity**: UBDroid is always up and running while the smartphone is running. Any uncaptured usage information will negatively affect analysis that will subsequently be performed on the data. UBDroid is automatically started and run at the background as a system service so that it cannot be stopped by the user or task killer applications.
- **Anonymity**: The system does not require personal information of the user. A user token generated from the device IMEI number is used for identification of user. The token is attached to the every request of the client to the server.
- **Energy efficiency**: The system is power efficient and does not reduce the battery life significantly.
    - Data collection: Data collection service does not run when the device is in sleep mode.
    - Data transmission: Collected data on the smartphone is transmitted to the server on a schedule to avoid frequently establishing and closing network connections.
- **Data consistency**: The data is collected from many clients whose system clocks may be different. Time is synchronized with the server when the client application starts.
- **Application categorization and statistics**: Application category, number of downloads and rating information is gathered from Google Play. Fetching the application information requires accessing to the web page of the application and crawling the page data. The application information can be fetched by the server application, while application usage data is stored, but that would

cause increasing transaction time and even failures because of timeout. It can be fetched by the client application, but it leads to increasing power consumption and cause different clients to fetch the same data. Instead of these approaches, a batch processor, which runs on the server, perform the application information retrieval on its own application context.

3.2 System Use Cases

UBDroid is data collection tool relies on server-client application. In UBDroid, there are two primary actors; user and administrator. There are also two supporting actors which are Background Services and Google Cloud Messaging. Users, participants of UBDroid, interact with client application whereas administrator interacts with server application via web client. As shown in Figure 1, users can register to the system, read messages, fill survey and send support message. Administrator can add survey, send message to users, manage user groups, display user status report, list support messages and start fetching application statistics. Background services run in background and are able to create notification, collect data and send collected data to the server. GCM transfers push notifications when messages sent from server to client.

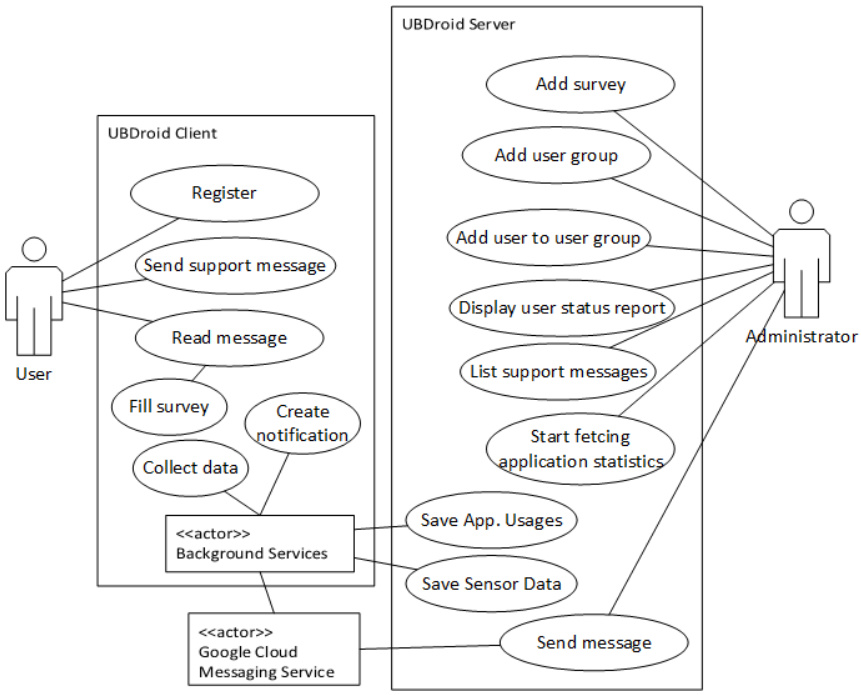

Figure 1 – UBDroid use case diagram

User should register to UBDroid. Data collection starts after registration. Administrator can creates and sends survey to registered users. User can fill survey and collected data can be uploaded. Main success scenario is covered in nine steps as follows.

Step 1: User registers to the system.

- User registers to GCM and GCM key is stored in local storage.
- Server client time is synchronized.
- Device information, installed application and GCM key are sent to server. Server persists user record in database and responds with user token.
- Background services; application usage collector service, sensor reader service, data uploader service and GCM handler service start to run.

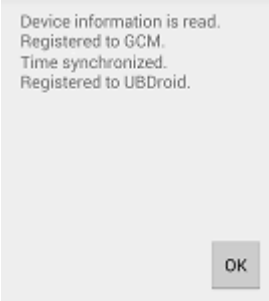

Figure 2 – Registration user interface

Step 2: Administrator creates survey

- Survey record is persisted to database.

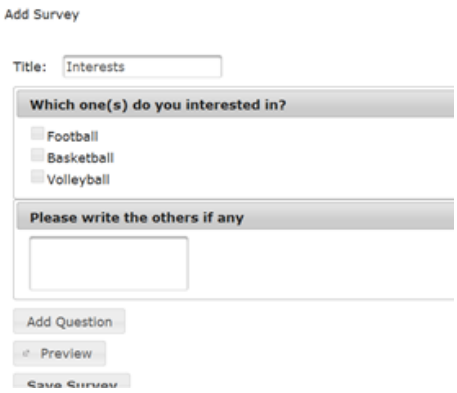

Figure 3 – Add survey user interface

Step 3: Administrator send message to user.

- Message record is created and persisted with survey link.
- Message metadata is send to GCM.
- UBDroid client receives push notification and stores message receive time.
- Notification is created.

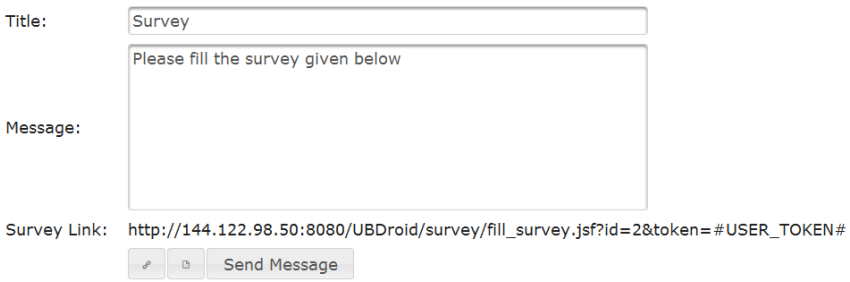

Figure 4 – Send message user interface

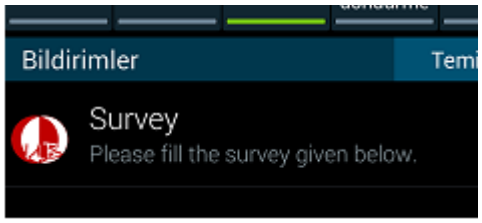

Figure 5 – Notification user interface

Step 4: User reads message.

- Message receive time is send to server and message detail is requested.
- Message read and receive time are updated and server responds with message details.
- Survey is displayed in message details.

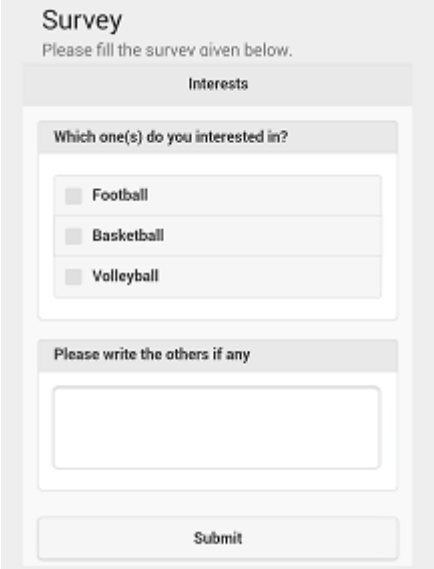

Figure 6 – Message user interface

Step 5: User fills survey.

- Survey answers record persisted to database and attached with user record.

Step 6: Administrator creates user group.

- User group record is persisted to database.

Figure 7 – Add user group user interface

Step 7: Administrator adds user to user group.

- User group record is persisted to database.

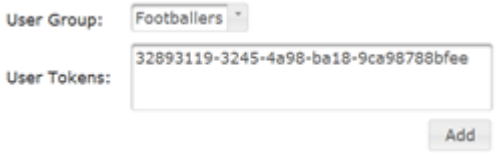

Figure 8 – Add user to user group interface

Step 8: UBDroid client service sends collected data to the server.

- Application usage records and sensor data records are sent to server. Server persists application usage and sensor data records to database if they are not persisted before.
- Server responds with success message and client removes the data from local storage.

Step 9: Administrator starts fetching application statistics.

- Application statistics are crawled from Google Play web page and persisted to database.

3.3 System Structure and Implementation Decisions

UBDroid is based on server-client architecture: Android smartphones are the clients, and the central application, that stores and processes all the data, is the server. Push notifications can be sent to clients through Google Cloud Messaging service. Overall system structure is shown in Figure 9.

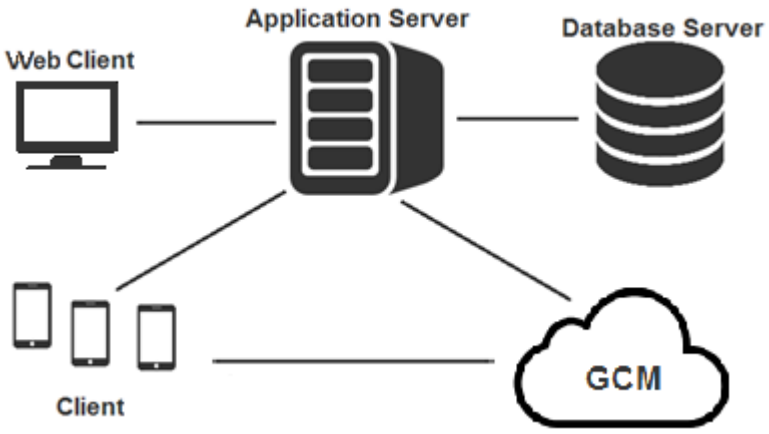

Figure 9 – System structure

It is very common approach to have layered application structure considering of separation of duties. Both server and client applications have the structures with three

layers as shown in Figure 10. Data layer is responsible of database operations such as saving, deleting, updating and querying. Business layer has core application functionality. Communication layer provides interaction with other systems. Interaction between server and client, server and GCM, server and web client, client and GCM are implemented in communication layer.

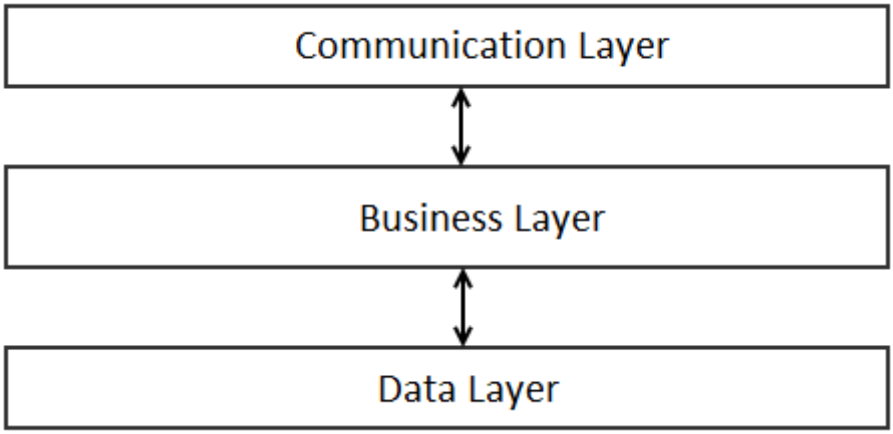

Figure 10 – Layered application architecture

3.3.1 Server Application Implementation Decisions

We have decided to run server application on *Apache Tomcat* web server and *MySQL* database server. We consider popularity, modularity, community support and documentation while deciding to use a technology, framework or library.

Server application has layered architecture as mentioned earlier. We have used *Hibernate* ORM library in data layer implementation. It provides mapping between Java classes and database tables and simplifies database operations. We have also avoided native SQL queries by using criteria mechanism of *Hibernate*.

In business layer implementation, we have integrated *Spring Framework* to our application. *Spring Framework* reduces coding efforts by providing dependency injection, aspect oriented programming support, modularity. It also helps to keep configuration stuff separated. We have also used *Spring Batch* library to process large amount of data. *Spring Batch* can access to database and help performing bulk

operations. We used *Spring Batch* for crawling data from Google Play. We have adapted *Selenium Tool* for crawling web page. *Selenium* is commonly used for automated tests for web applications, so it has advanced features for crawling.

Communication between applications is done by REST services. REST services allow data to be attached to HTTP request in JSON format. This requires making JSON to object and object to JSON conversion. At server application we have created REST services with *Spring MVC* Framework and integrate *Jackson* library for JSON conversion.

3.3.2 Client Application Implementation Decisions

Client application runs on Android powered smartphones. It has also three layers; data layer, business layer and communication layer. Android natively supports *SQLite* database for local storage. We implemented data layer by using *ORM Lite* library for database operations. It provides mapping between java classes and database tables like *Hibernate*, but it is a lightweight library that is suitable for mobile applications. We do not need library or framework at business layer, and implement business logic with native Android services and activities. In order to interact with server, we used REST client library called *Retrofit*. We integrated *Retrofit* with *Jackson* library for JSON conversion.

3.3.3 Web Client Application Implementation Decisions

Web client application relies on model-view-controller architectural pattern as shown in Figure 11. We implemented web client application by using *Primefaces Framework*. It is MVC implementation based on JSF framework. It has a large set of user interface components which simplifies building web applications.

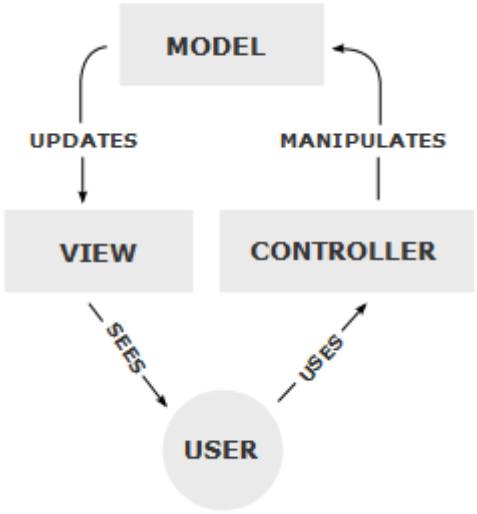

Figure 11 – Model-View-Controller pattern [24]

3.4 Server Application Design

Main focus of the UBDroid server is processing and storing the data collected by the clients. There are five modules in server application; user module, survey module, message module, data collection module and batch module.

3.4.1 User Module

Entity

User module entities are user, device and user group as shown in Figure 12 – User module ER diagram.

Figure 12 – User module ER diagram

Functionality

User module has the functionalities of; registration, user group management and user summary creation.

- Registration: IMEI, device brand/model, Android version, list of installed applications and Google Cloud Messaging key are taken from user. User record is created based on the device identifier. Device identifier is a hash value generated by SHA-256 algorithm from device IMEI appended by a random salt value, so that IMEI cannot be extracted from the device identifier by decryption or brute force.

  When the registration is successful, user token and sequence numbers of user data is returned to the client application. User token is used for identification of the user. If the user is already registered, existing user token for device identifier is returned instead of creating new user. Data sequence numbers are used for data synchronization, they are returned in order to handle data synchronization when the client application is re-installed.
- User Group Management:

    - Add user group: A new user group is created with the user provided name and description.
    - Add the user to a user group: Users, whose tokens are given, are added to a specific user group.
- User summary creation: All users' current status are listed. The summary information includes enrolled user groups, filled and unfilled surveys, read and unread messages, latest data transfer time for each user. The summary information is helpful to detect inactive users.

3.4.2 Survey Module

Entity

Survey module entities are survey, question, choice, survey answers and answer as shown in Figure 13. There are four types of questions; comment, short comment, single selection and multiple selections. There are four types of answers for each type of questions. Lastly, there are two types of choices; comment choice and selection choice.

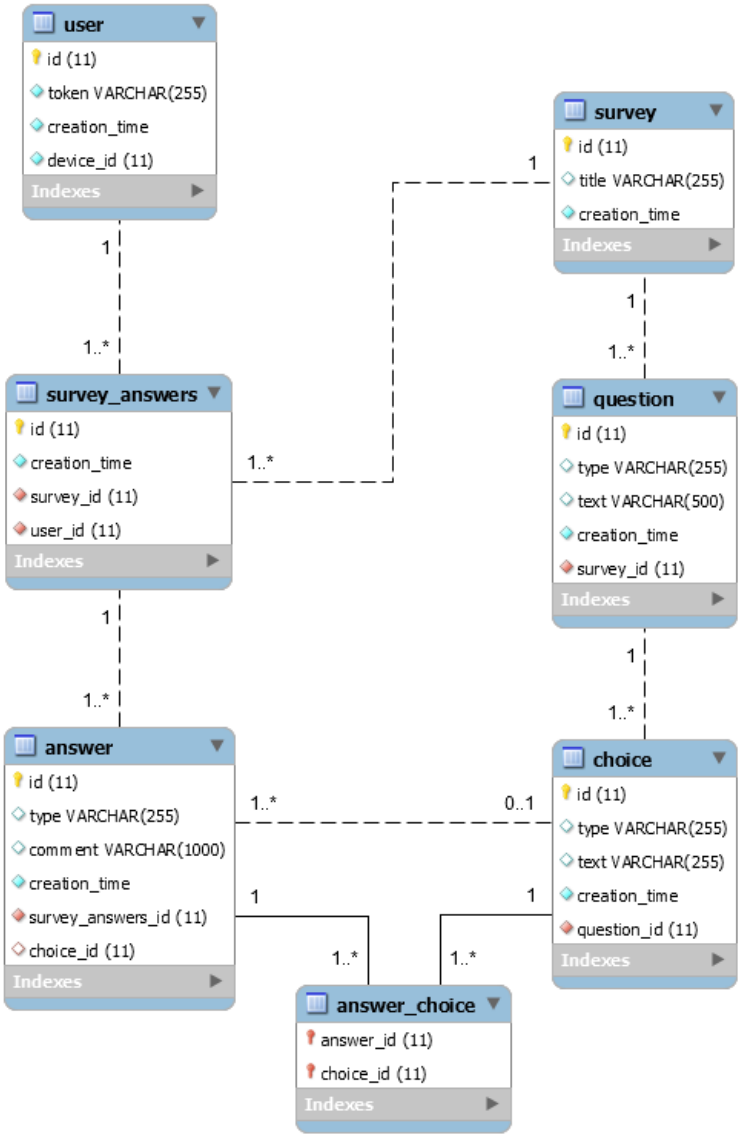

Figure 13 – Survey module ER diagram

Functionality

Survey module allows participatory data collection. Surveys can be added and filled dynamically.

- Add survey: A survey with given questions is persisted to the database.
- Get survey: The survey with the given id is returned.
- Get surveys: List of all the surveys is returned.

- Add survey answers: Survey answers filled by the user are persisted to the database.

3.4.3 Message Module

Entity

Message module entities are message and support message as shown in Figure 13 – Survey module ER diagramFigure 14. There are three types of messages; text, html and command. Text message is used for ordinary messages. Html message consist of an URL to display the web page in the message. Command message is designed to trigger actions on the client application.

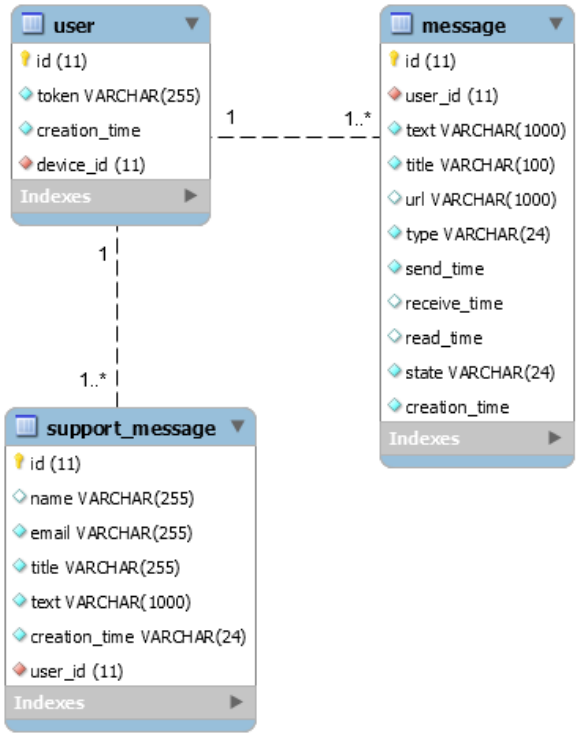

Figure 14 – Message module ER diagram

Functionality

Message module is responsible for sending messages to the users and receiving support message from the users.

- Send message: Message is created and persisted to the database for given users. Also, push notifications are sent by using Google Cloud Messaging service in order to notify the users.
- Send command message: Command message is created and persisted to the database, and then push notification is sent. Command message triggers sensor data collection or data transfer on the client application.
- Get messages: Returns all text and html messages of a given user which are not marked as deleted.
- Read message: Returns the message by given message id. In addition, receive time and read time of the message are updated.
- Delete message: Marks the message as deleted. Message is still kept in the database.
- Save support message: Support message is persisted to the database.
- Get support messages: Returns the list of all support messages.

3.4.4 Data Collection Module

Entity

Data collection module entities are sensor data, application usage, application information and installed application as shown in Figure 15Figure 15 – Data collection module ER diagram. Application usage information includes package name and activity name of the application. Package name is identifier for the application, and activity name is used for each page in the application.

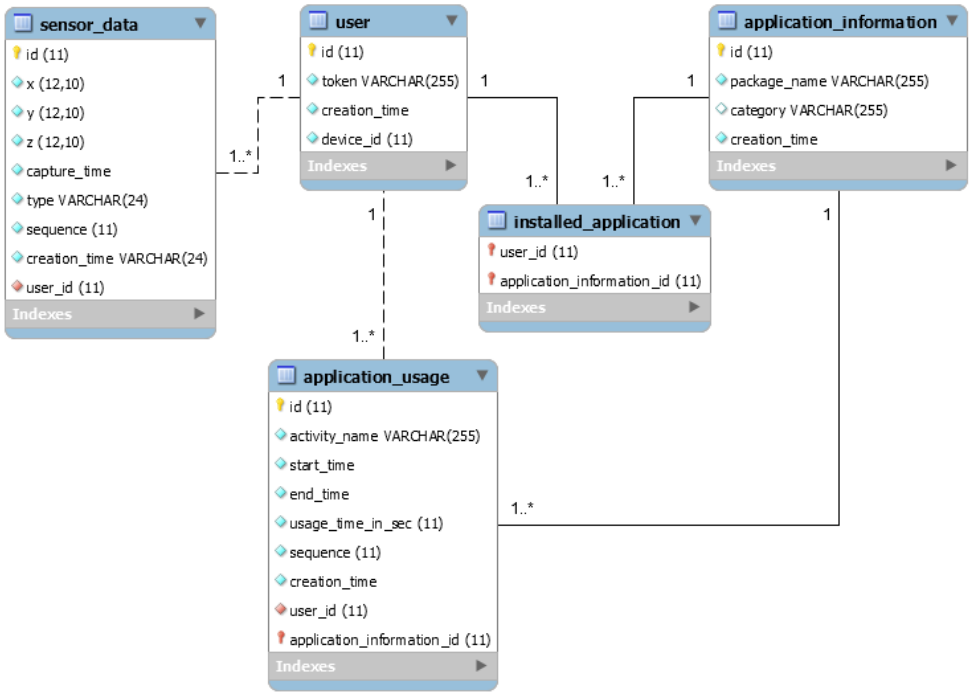

Figure 15 – Data collection module ER diagram

Functionality

The main responsibility of the data collection module is synchronization of collected application usage data and sensor data.

- Save application usage: Application usage data is persisted to the database if not already persisted.
- Save sensor data: Sensor data is persisted to the database if not already persisted.
- Save installed application: Application information is persisted to the database as installed applications of user.

Application usage or sensor data records should be uniquely identified so that no record should be duplicated or missed. Therefore, every application usage or sensor data record is related to user and a sequence number. The synchronization algorithm is described in pseudocode below:

```
find user by token
for each data
        if sequence of data > max sequence for the user in database
                relate data with user
                persist data to database
return success message
```

3.4.5 Batch Module

Entity

Batch module entities are application information and application statistics as shown in Figure 16.

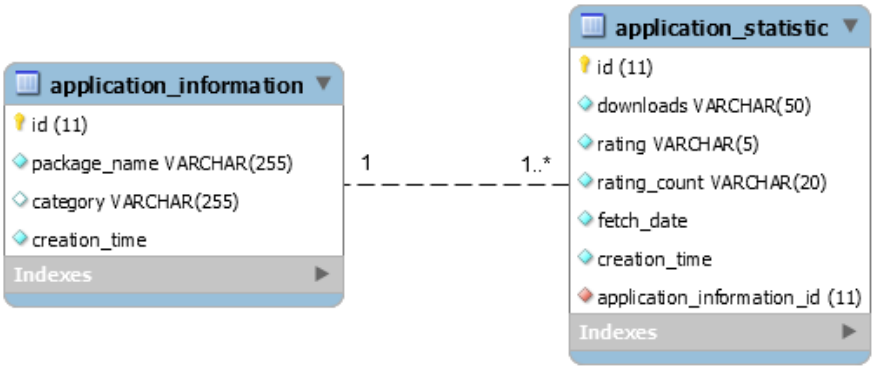

Figure 16 – Batch module ER diagram

Functionality

Batch module is responsible for gathering application information: category, number of downloads and rating. Batch module is a part of the server application, but it has its own context for time consuming jobs. Unlike request-response mechanism of server applications, a batch job does not make user wait until it is completed. It continues to work in background. Batch job is done in three phases; read, process and write as shown in Figure 17 – Batch module structure . When a job is started, all unique package names are gathered from the database. For each unique package name, batch executes the following procedure:

- *ItemReader* fetches *ApplicationInformation*.

- *ItemProcessor* extracts category, number of downloads and rating from Google Play.
- *ItemWriter* updates *ApplicationInformation* and inserts *ApplicationStatistic* to the database.

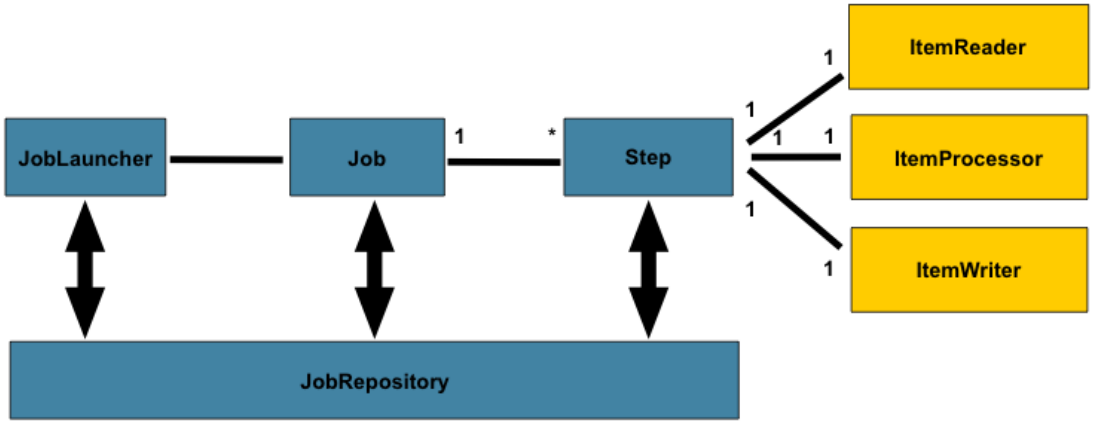

Figure 17 – Batch module structure [25]

3.5 Web Client Application Design

Web client is used for administration. Web client interacts with four modules of the server application; user module, survey module, message module and batch module.

3.5.1 User Module

User module allows adding a new user group, adding a user to a user group and listing status of the users.

- Add user group: Name and description is sent to the server to create a new user group.
- Add a user to a user group: List of user tokens with the selected user group is sent to the server.
- User status: Enrolled user groups, filled and unfilled surveys, read and unread messages, latest data transfer time are fetched from the server and listed in a data grid.

3.5.2 Survey Module

Survey module of the web client allows adding and listing surveys.

- Add a survey: A new survey can be added after all questions of the survey are created. Four types of questions can be created; comment, short comment, single selection and multiple selection. Questions can be ordered, deleted, copied or edited. Survey is, also, previewed before it is saved.
- List surveys: All surveys are listed in a data grid.

3.5.3 Message Module

Message module is used to send message to the users and list support messages.

- Send message to a user: A message is created with a title, text and, optionally, a survey for the selected user.
- Send message to a user group: A message is created with a title, text and, optionally, a survey for all the users in the selected user group.
- Send command message: Command message is created with selected type to trigger transferring usage data, reading sensor values or getting the location. If reading sensor data is selected, then type of sensors to be read (Accelerometer, Gravity, Gyroscope, Magnetic Field), number of reads and read interval must be selected.
- List support messages: Support messages are listed in a data grid.

3.5.4 Batch Module

Batch operations run in the background, but batch module of the web client allows monitoring the operations.

- Fetch application statistics: Starts fetching applications statistics from Google Play.
- List batch job execution status: Start time, end time and status of batch operation execution are listed in data grid.

3.6 Client Application Design

UBDroid client runs on Android smartphones. It has the core responsibility of data collection. Application usage and sensor data are collected without user involvement, but the client allows collecting participatory data by using messages and surveys. Client consists of three modules; user module, data collection module and message module.

3.6.1 User Module

User module is responsible for the registration process. No user input is required to register UBDroid, but following three steps executed without user notification, these steps are also shown in Figure 18.

- Google Cloud Messaging service registration: UBDroid uses GCM service for sending and receiving push notifications. At this step, GCM service returns the unique gcm_key.
- Time synchronization: Each client collects data and attaches time information by using its own system clock. Since data is centralized, ideally every client should have the same clock. Instead of synchronizing the time at every data upload, time difference between the client and the server is calculated before the registration. By this way, the data is attached with the server time while it is collected. If time is changed by user, UBDroid re-synchronizes the time. Unfortunately, usage data recorded with old time information until the synchronization is done. Time synchronization is performed based on Cristian's Algorithm [26]. Client sends request to server, and server responds with its current time. Client sets the time by adding half of round trip time of the request. The algorithm removes the effect of network delay where request and response delays are equal. If the delays are not equal, the maximum time difference between client and server will be 7.5 seconds since client application has 15 seconds of timeout duration.
- UBDroid registration: IMEI, device manufacturer/brand/model, Android version, installed applications, gcm_key information are sent to the server.

The server returns user token to the client, indicating that the registration was successful. User token acts as an identity. It is stored in the local storage and attached to every request to the server.

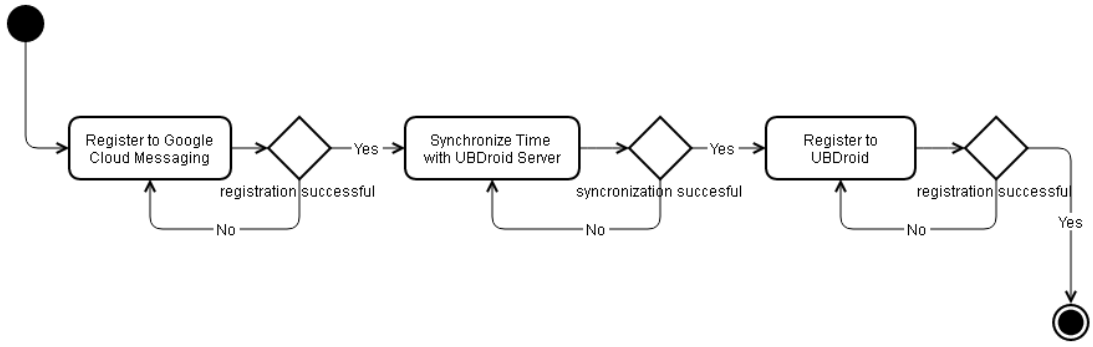

Figure 18 – Registration activity diagram

3.6.2 Data Collection Module

The functionalities of the data collection module are extracting, storing, and synchronizing application usage and sensor data. Since these operations are done without user interference, they are run as background services. There are three background services in UBDroid client application as follows.

Application Usage Collector Service

Application usage collector service is the executor of application usage data collection. As class diagram presented in Figure 20, application usage information is gathered by watching foreground application changes. Android Framework does not allow 3rd party applications to handle application change event, but provides functionality to get foreground application information. The event mechanism is achieved by polling. *ApplicationChangePublisher* is responsible to poll foreground application by an interval. Longer polling interval causes lower sensitivity on application running time calculation. In addition, the applications used less than the polling interval might be skipped as illustrated in Figure 19. On the other hand, short polling interval causes increasing number of calculations which leads to increased battery consumption. We decide polling interval of 1 second by assuming an

application, used less than a second, is not actually used and the usage can be skipped.

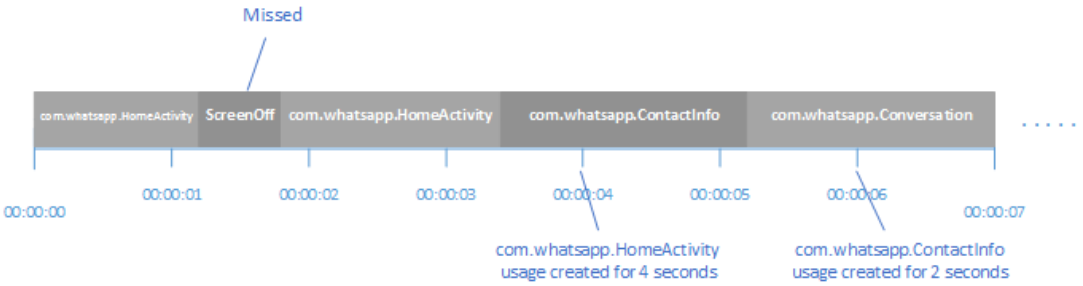

Figure 19 – Application usage collection timeline

ApplicationChangeEvent is published to *EventBus w*henever foreground application change is detected. *ApplicationUsageCollectorService* is registered for *ApplicationChangeEvent*. When the event is handled, application usage information is created and persisted to the local storage. *ApplicationChangePublisher* stops polling when the device screen turns off, and restarts polling when the device screen is back on. Screen On/Off state can be listened by Android Framework, so *ScreenOffListener* is not registered to *EventBus*.

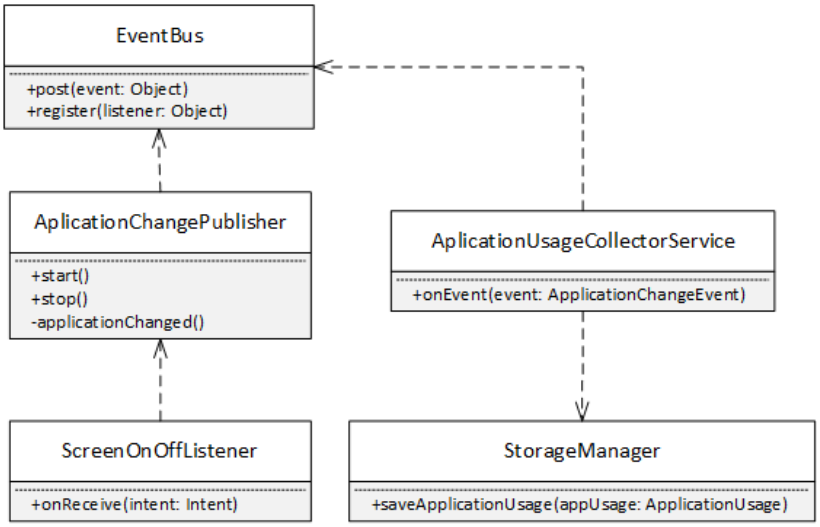

Figure 20 – Application usage collection class diagram

Sensor Reader Service

As shown in Figure 21 – Sensor data collection class diagram, sensor reader service runs in conjunction with *EventBus* and *StorageManager* like the application usage

collector service, but sensors are not continuously read to prevent battery drain. The service is able to read accelerometer, gravity, gyroscope and magnetic field sensors. Sensor reading is triggered by *SensorReadEvent* which is fired by push notification. *SensorReadEventListener* reads sensor value at given intervals and count defined in *SensorReadEvent*. Sensors can be read even if the device is in sleep mode by acquiring wake lock. Wake lock is released whenever sensor reading is done, so large number of reads and longer reading interval will increase battery consumption. Unlike application usage data, sensor data is sent to the server just after sensor reading is done to give possibility of analyzing collected data instantly. If sensor data transfer fails, it stays in the local storage until data uploader service synchronizes all the collected data.

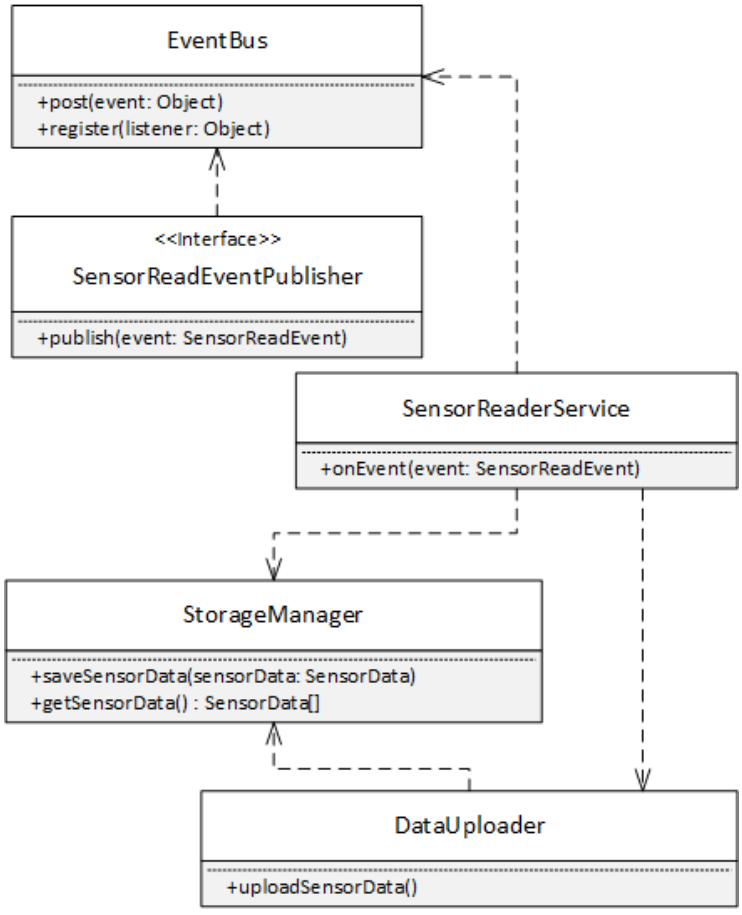

Figure 21 – Sensor data collection class diagram

Data Uploader Service

UBDroid client transmits collected data via data uploader service. Data upload is performed on a daily schedule in order to decrease battery consumption by avoiding constant network connection. *DataUploadScheduler* fires *DataUploadEvent* to *EventBus* once a day. *DataUploaderService* is triggered by data upload event. It fetches all collected data from the local storage and sends to the server via *UBDroidRemoteService* interface. If there is no network connection during a data upload attempt, then *DataUploadScheduler* fires *DataUploadEvent* again when network connection is enabled.

The collected data is sent to the server by dividing into parts as it might be too large to process efficiently at a single request. After a part of the data is persisted to the database, the server returns success message to the client. Then, the client deletes this part from the local storage of the device and sends the next part of the data to the server. If the client misses server's success message, the client will resend the part at next synchronization. However, server will be aware that the records were already persisted, and does not duplicate the records.

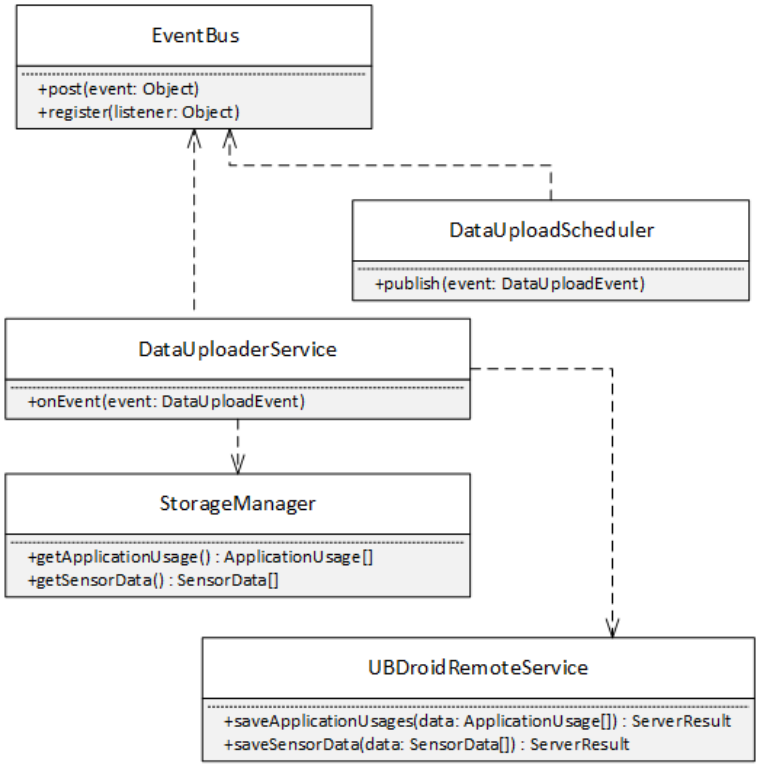

Figure 22 – Data upload class diagram

3.6.3 Message Module

Message module manages operations such as listing messages, reading a message and deleting a message. It is also integrated with GCM and it handles command messages sent by the server.

Push notifications are handled by *GcmHandler*. If a command message arrives to the handler, then the corresponding event is fired. *GcmHandler* communicates with services via *EventBus* as shown in Figure 23. When a message arrives to the handler, a system notification is created to notify the user. Also, the arrival time of the message is stored in the local storage. When the message is read by user, arrival time and read time of the message are updated.

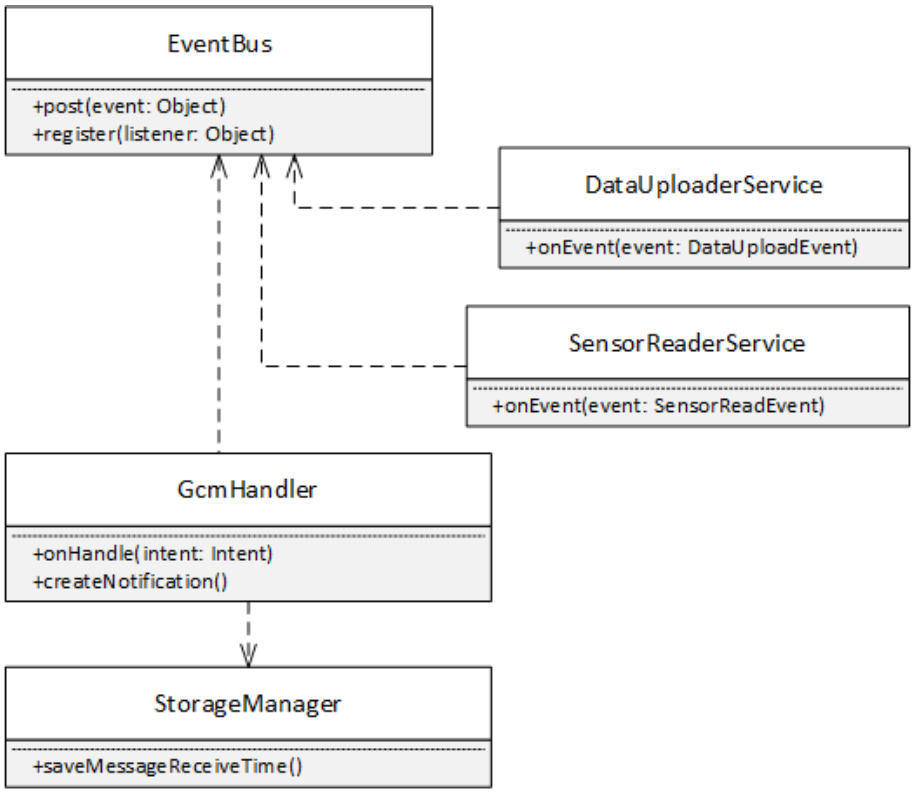

Figure 23 – Message module class diagram

CHAPTER 4

SYSTEM PERFORMANCE TEST AND RESULTS

Using system resources efficiently plays a critical role in mobile sensing applications. The data collection process must have high accuracy, so that any sample should not be missed. On the other hand, the application should not have a significant negative impact on the overall performance of the mobile device. It should have little extra CPU usage overhead not to have any adverse effect on the operation of the other running processes. It should not have any significant effect on battery drain not to have a negative impact on the user experience. Considering battery efficiency, network connections should be used efficiently as well. Memory should be used efficiently in order to make it possible to run the application able on lower end mobile devices.

The main focus of UBDroid client is application usage data collection. Application usage data collector service continuously runs in the background whenever mobile device is in use and data collection is not performed when device is in sleep mode. Besides, UBDroid allows reading sensor values for a period of time. However, sensor values can be read even if the device is in sleep mode. All the collected data is sent to the remote server on a daily schedule. UBDroid is tested for the specific cases in terms of CPU, memory, disk, network and battery usage.

The main purpose of UBDroid server is data centralization by handling data transfer requests from clients. The server is tested by the maximum number of requests that can be handled the same time. It plays significant role in determining how much users can use UBDroid.

4.1 UBDroid Client Test

4.1.1 Test Setup

UBDroid client is designed to run continuously in background. Application usage data is stored in local storage until it is transferred to the server. The amount of collected data affects CPU utilization, battery and network usage while the data is transferred. We mainly performed tests based on application usage data since it is the big part of collected data. Before testing client application, we collect data from 12 users for 3 days. We saw that screen is turned 20% of time on average whereas 11% is minimum, 35% maximum. The most active user produced 822 application usage records per day and the most passive user was 290. Average number of application usage records created per day was 478. In tests, we assume that average user creates 500 application usage records and keeps screen on for 5 hours per day. Sensor data collection is triggered by push notification. The number of sensor reads is decided by administrator. We assume that 120 sensor data records are created for each four sensors. We performed data transfer tests with 500, 1000, 1500 and 2000 number of application usage records. We used 3 smartphones for testing as listed in Table 3.

Table 3 – Tested smartphones

Device	CPU	Memory	Battery	Android Version
Samsung N900	Quad-core 1.9 GHz Cortex-A15 and Quad-core 1.3 GHz Cortex-A7	3GB	12.16Wh (43776J)	4.4.2
Samsung I8190	Dual-core 1 GHz Cortex-A9	1GB	5,7Wh (20520J)	4.1.2
Samsung I997	Single-core 1.2 GHz Cortex-A8	512MB	6,65Wh (23940J)	4.2.2

4.1.2 Test Results

We performed tests for inferring memory, disk, network, battery and CPU usage.

Memory Usage

Memory usage is monitored by connecting smartphone to Dalvik Debug Monitor Server (DDMS) [27]. We performed the tests after application user interface is loaded and all background services are running. Maximum memory usages are captured while different number of application usage data is transferred. Memory usage in idle is captured when there is no data transfer. As shown in Table 4, memory usage varies between devices since Android manages memory differently according to amount of memory that device has. In order to include results in lower memory device, we also test UBDroid client on virtual device, which simulates Android device on computer, with 256MB memory. As the results show, the maximum memory usage of UBDroid client is around 19MB and it is able to run on low memory devices.

Table 4 – Memory usage (MB)

Device/Number of Records	0 (Idle)	500	1000	1500	2000
Samsung N900	16.82	17.24	17.65	18.13	18.58
Samsung I8190	9.80	10.16	10.34	10.55	10.81
Samsung I997	7.72	8.05	8.36	8.66	8.95
Virtual device	3.05	3.32	3.60	3.90	4.19

Disk Usage

Application usage data is stored in Android's built-in SQLite database. The amount of used local storage is extracted from the system settings of the device. We examined how much local storage is used while application usage data is collected as shown in Table 5. Data is removed from local storage when it is transferred to the server. As a result, if data is collected for a week without transferring to the server, local storage will be used less than 420KB for an average user.

Table 5 – Disk usage (KB)

Device/Number of Records	500	1000	1500	2000
Samsung N900	60	104	160	200
Samsung I8190	60	116	172	232
Samsung I997	60	114	170	228

Network Usage

Network usage is monitored by using DDMS. As shown in Table 6, all devices produced very similar results. The data is converted to a standard JSON format and appended to request body, so the sizes of requests are exactly same. The small differences occur because of network situation that causes resending some of network packages. Assuming that average user produces 500 application usage records per day, the total size of data transfer will be 2.88MB over a month.

Table 6 – Network usage (KB)

Device/Number of Records		500	1000	1500	2000
Samsung N900	Tx	90.61	181.74	271.27	362.76
	Rx	7.44	14.86	21.66	28.75
Samsung I8190	Tx	89.39	183.84	278.45	360.32
	Rx	6.78	13.88	21.54	28.13
Samsung I997	Tx	89.66	182.32	276.85	360.41
	Rx	7.18	13.65	21.80	28.64

Battery Usage

We performed battery usage test in order to estimate how much battery is consumed over a day. Since Android does not provide detailed battery usage statistics, we used PowerTutor [28] for the measurements. It calculates consumed energy for each application. We aimed to determine how much energy is consumed over a day by using UBDroid We separately tested application usage collection, sensor data collection and data transfer for each.

Aa average user creates 500 application user records. Since device screen is on for 5 hours, foreground application is polled 18000 times per day. Application usage is

tested with 500, 1000, 1500 and 2000 records. Sensor data is test with 60, 120, 180 and 240 records for each four sensors available in UBDroid. Battery consumption results while polling foreground application is shown in **Error! Reference source not found.**. Application usage collection and transfer usages are shown in Table 8 and Table 9. Sensor data collection and transfer results are shown in Table 10 and Table 11.

Table 7 – Battery usage on foreground application polling (J)

Device	Energy Consumed
Samsung N900	182.5
Samsung I8190	203.1
Samsung I997	218.9

Table 8 - Battery usage on application usage collection (J)

Device/Number of Records	500	1000	1500	2000
Samsung N900	6.1	10.2	14.6	17.9
Samsung I8190	6.9	11.9	16.0	20.7
Samsung I997	7.5	12.8	16.9	22.3

Table 9 – Battery usage on application usage transfer (J)

Device/Number of Records	500	1000	1500	2000
Samsung N900	17.1	25.8	35.8	45.2
Samsung I8190	16.5	30.9	47.1	57.8
Samsung I997	19.2	33.2	49.3	63.6

Table 10 – Battery usage on sensor usage data collection (J)

Device/Number of Records	60	120	180	240
Samsung N900	4.4	7.9	11.6	15.7
Samsung I8190	5.8	10.0	14.1	18.8
Samsung I997	6.1	10.5	14.8	19.2

Table 11 – Battery usage on sensor data transfer (J)

Device/Number of Records	60	120	180	240
Samsung N900	10.1	19.6	31.3	40.5
Samsung I8190	10.9	21.0	33.1	42.6

Samsung I997	12.4	22.8	34.9	45.2

Overall battery consumption is calculated by summing values for 500 application usage records and 60 sensor data records as listed in Table 12. UBDroid client consumed around 0.5% battery of high-end smartphone and 1% of mid-range smartphone over a day. Note that the battery consumption is calculated for background service of UBDroid client, higher consumptions occur when a user reads messages and fills surveys.

Table 12 – Overall battery usage for average user per day (J)

Device	Consumption	Consumption / Battery Capacity
Samsung N900	220.2	0.5%
Samsung I8190	243.2	1.2%
Samsung I997	264.1	1.1%

CPU Usage

There is no built-in tool for logging CPU usage in Android operating system or Android Development Toolkit. We calculated CPU usage by using 3C Toolbox [29]. 3C Toolbox gives the process time that application consumes CPU. It also calculates usage ratio which is process time divided by application running time. We performed test based on average user. We calculated CPU usage time by polling foreground application 18000 times and creating 500 application usage records. In addition, the collected data is transferred to the server. CPU usage results are calculated by the total usage time divided by 5 hours which is shown in Table 13.

Table 13 – CPU usage

Device	CPU usage
Samsung N900	0.5%
Samsung I8190	0.8%
Samsung I997	1.0%

4.2 UBDroid Server Test

4.2.1 Test Setup

UBDroid server centralizes the data collected from clients. We performed load tests on server application while application usage data is collected from multiple users. The tests were performed on mid-end laptop specified in Table 14. JMeter was used for server tests [30]. It is performance measurement tool for web applications. We performed tests with 10, 20, 30, 40 and 50 concurrent users who upload application usage data. Each request contains 50 application usage records to be persisted as UBDroid client behaves. We run the test continuously; when a request is done, new one is started.

Table 14 – Test server specifications

Processor	Intel Core i5 2450M
Memory	8GB (1GB is dedicated to the server)
Hard Disk Drive	Western Digital WD7500BPVT
Operating System	Windows 7 Home Basic 64-Bit

4.2.1 Test Results

As expected behavior, servers should respond to client in short time without error. However, response time increases under heavy load. Load tests are intended to determine the critical limit that server started to return failed response. The acceptable response time is taken 5 seconds at ideal cases. We set the timeout duration as 15 seconds for client application. According to our test results as shown in Figure 24, 10 and 20, concurrent requests are successfully handled. 30 concurrent requests exceeds acceptable limit. 40 concurrent requests mostly exceed timeout duration. The server started to return unsuccessful results where is the number of concurrent requests is 50. The test results showed that UBDroid server can handle only 20 concurrent requests on a mid-end laptop. The results mean that UBDroid, which is running on a mid-end laptop, can serve 20 users at worst case scenario.

Instead of transferring collected data at the same time, there must be scheduling mechanism to ensure that users upload the data at different time of day.

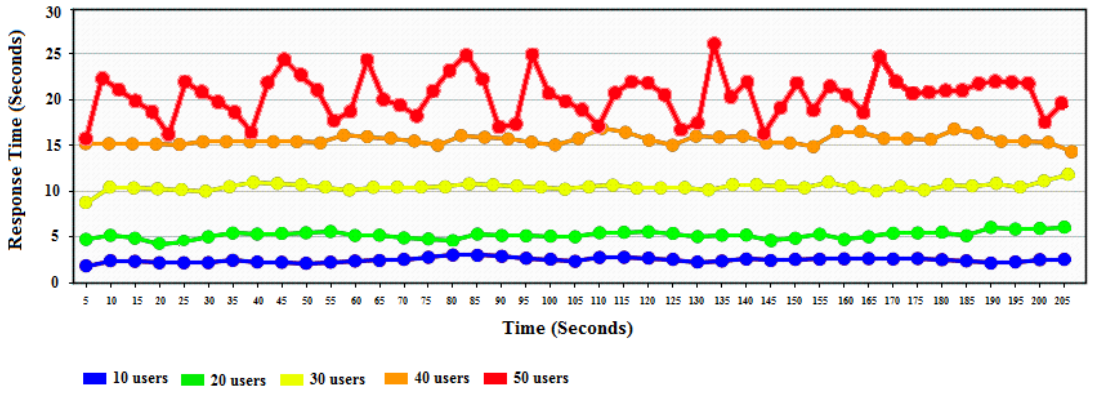

Figure 24 – Response time graph

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

In this study, we develop a tool to create valuable database for mobile user behavior analysis. We regard usage of mobile device resources for data collection and synchronization. We performed tests on UBDroid client that indicates no significant decrease in performance of test device. Tests also show that UBDroid client consumes battery and use network connections efficiently. In addition to application usage and sensor data collection, UBDroid has online survey capability. Key features of UBDroid are listed as follows:

- Collecting application usage data.
- Fetching application statistics from Google Play.
- Collecting sensor (accelerometer, gravity, gyroscope, magnetic field) data in a period of time.
- Collecting current location.
- Creating and performing online surveys.
- Informing users via messages.
- Triggering sensor data collection, location collection and collected data synchronization on clients via push notifications.
- Allowing administration of the system via web client.

UBDroid is intended to be used in different research contexts related to Android smartphone users. UBDroid gets installed applications, tracks running applications, fetches their Google Play statistics, collects sensor data, sends messages to users and performs online surveys. By using application usage information, it is possible to develop application recommendation system as the study of AppJoy [15] shows.

UBDroid also gives opportunity to analyze user activities such as the time of day when the device is in use; the time of message received and read. By applying InterruptMe model [14], opportune moments of users can be inferred. In other words, most responsive moments or users can be analyzed by using the data collected by UBDroid. In addition, MoodScope [13] predicts user mood according to application usage data. UBDroid creates valuable database for user behavior analysis that can be used in different contexts.

5.2 Limitations

UBDroid client runs on Android smartphone. The minimum required device specifications listed as follows:

- Android version must be 2.3 or above.
- Minimum amount of RAM must be 256MB.
- Minimum amount of free disk space must be 20MB.
- Minimum screen resolution must be 480x320 in order to display surveys properly.
- Google Play Services must be installed for enabling push notifications and determining current location.

Application usage information includes package name and activity name of the application. Activity name cannot be determined at devices that run Android 5.0 or above since the current APIs dropped support for getting running activities. [31]

UBDroid collects data anonymously since users give importance on privacy. In UBDroid, data collection is not limited to experimental environment or specific time interval. Sensor data could be fetched in arbitrary times whenever smartphone has internet connection. By considering this, microphone is not included in sensors to be used because it may carry sensitive information for users.

UBDroid is designed to consume smartphone resources efficiently. Sensor data collection is not continuously performed considering battery consumption. Instead of

this, sensor data is collected for a period of time which is triggered by push notifications.

UBDroid server runs on Apache Tomcat 8 application server and MySQL 5 database server. The server can run on a system with 1GB memory and 500MB free disk space. However, the required specifications vary depending on number of users and the duration of data collection. In addition, server application requires a GCM enabled Google account for push notifications.

5.3 Future Work

UBDroid is a data collection tool for mobile phone users. Its administration can be done by using web client. Some functions should be triggered by administrator manually. For instance, fetching application statistics needs to be triggered by web client. Sensor data collection is also started manually. There might be scheduling rules that trigger the functions, so UBDroid could be more automated tool.

Collected data is transferred from clients to the server daily. The first transfer is done one day after registration. If many users are registered at the same time, then all of them will try to transfer data at the same time. This situation may result in overload on server application according to the number of users. An advanced scheduler mechanism can be designed which reserves different time of day for each user. Therefore, it can be ensured that overload is prevented.

APPENDICES

Appendix A: Web Client Application User Interfaces

Web application user interfaces, which are used for administration, listed in this appendix. User module interfaces for listing user status and managing user groups are shown in Figure 25, Figure 26 and Figure 27. Survey adding and listing interfaces are shown in Figure 28, Figure 29 and Figure 30. Message module interfaces are shown in Figure 31, Figure 32 and Figure 33. Batch module interface to start and monitor batch jobs is shown in Figure 34.

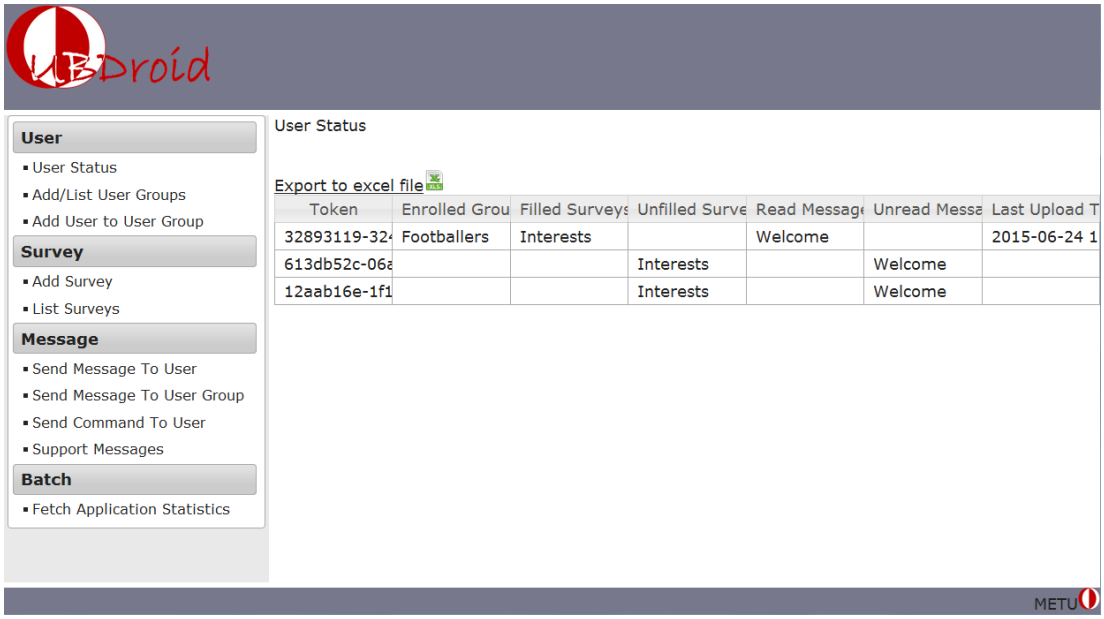

Figure 25 – User status web interface

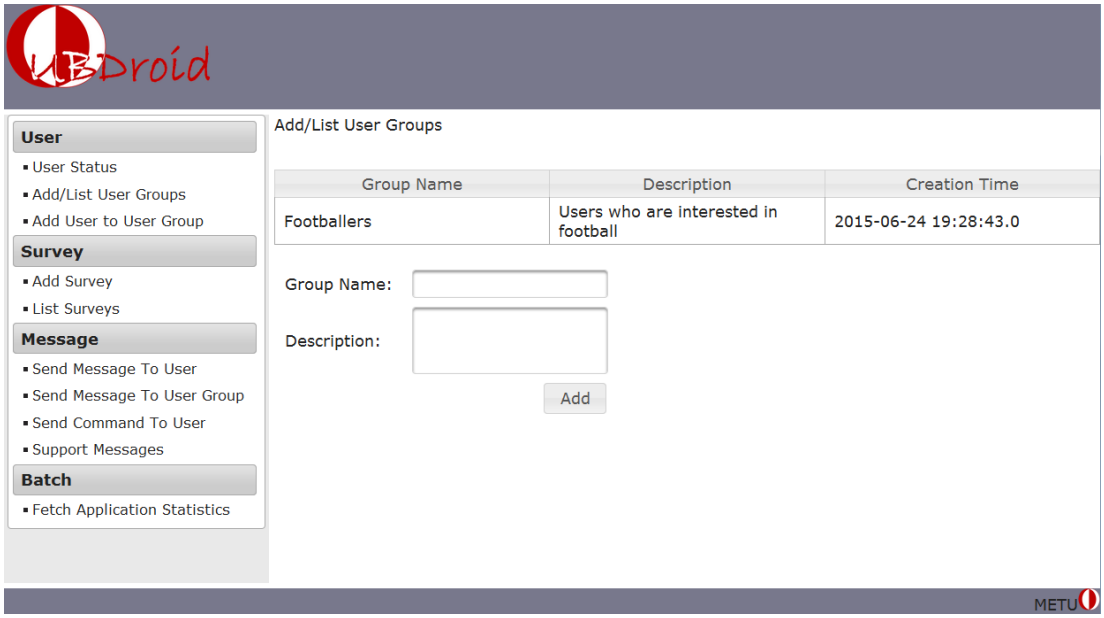

Figure 26 – Add list user group web interface

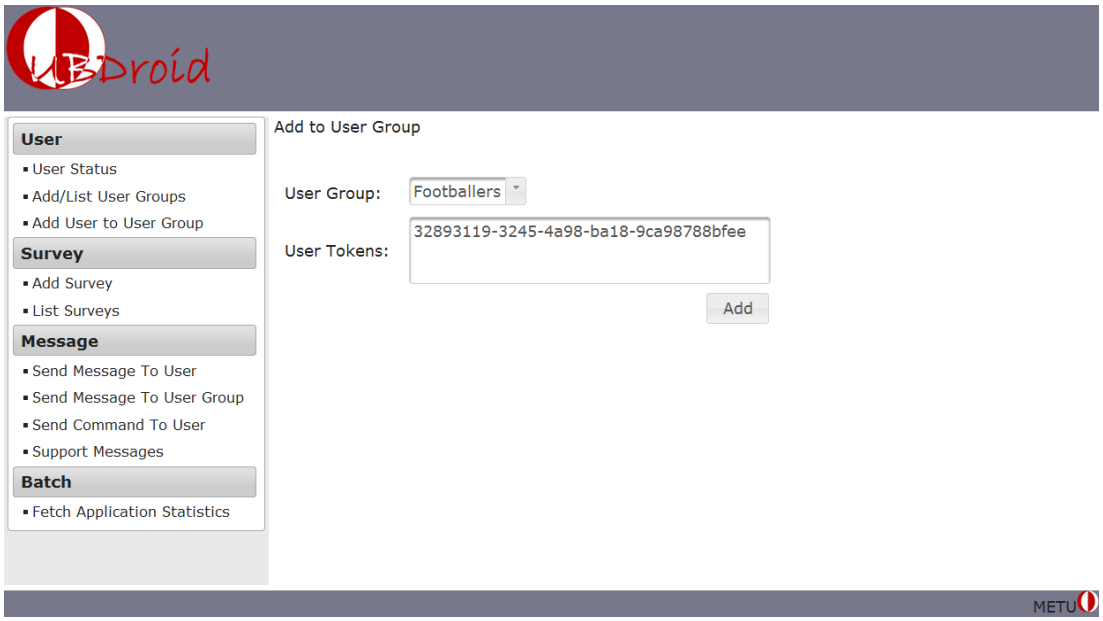

Figure 27 – Add user to user group web interface

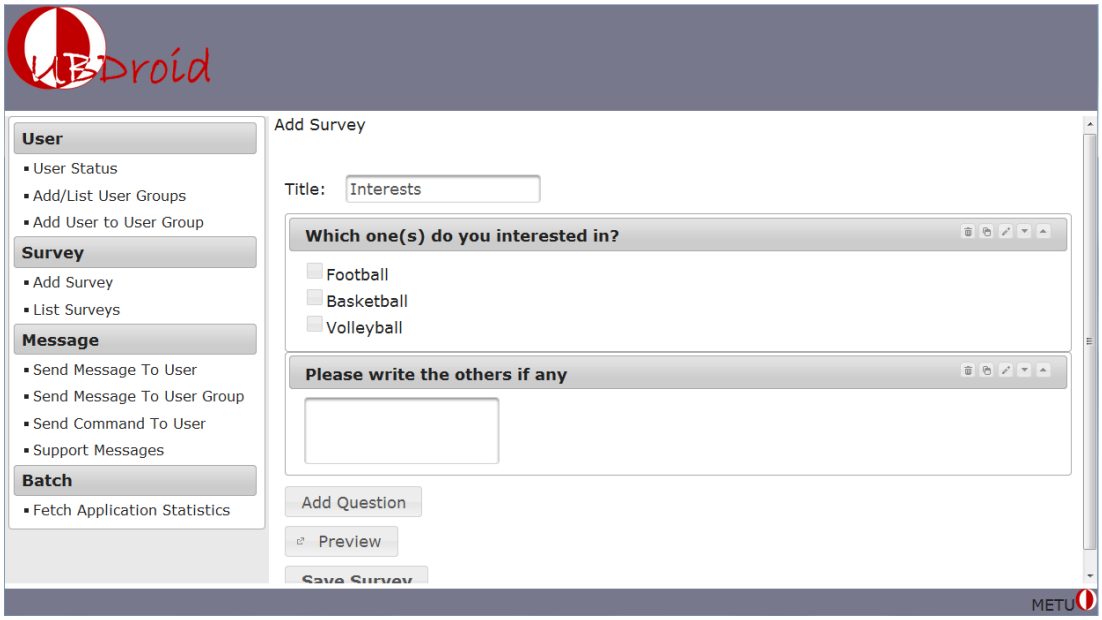

Figure 28 – Add survey web interface

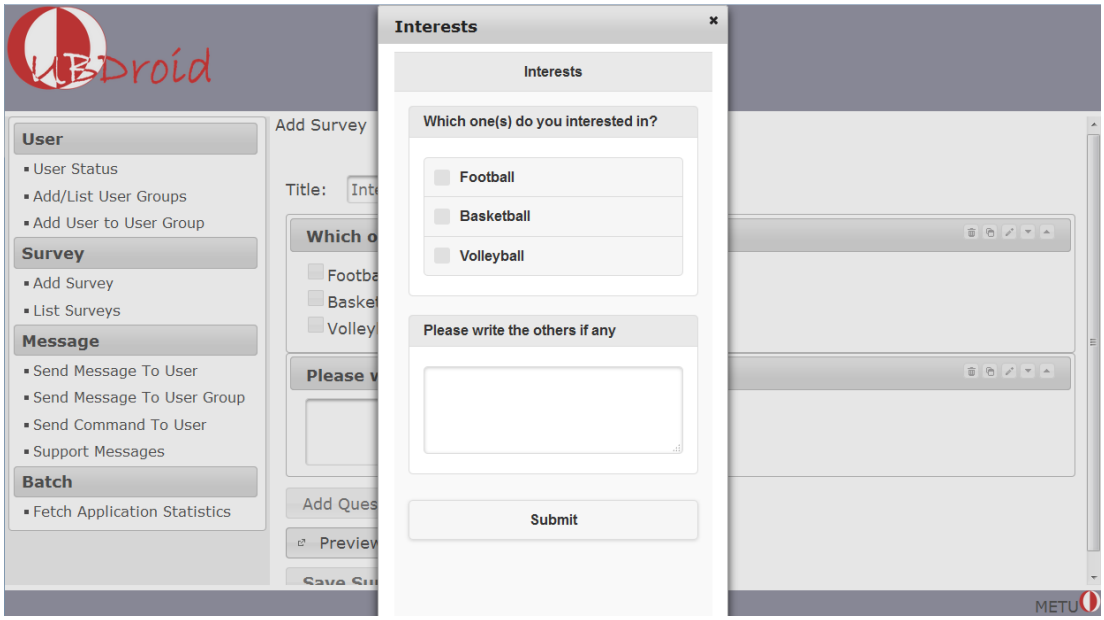

Figure 29 – Preview survey web interface

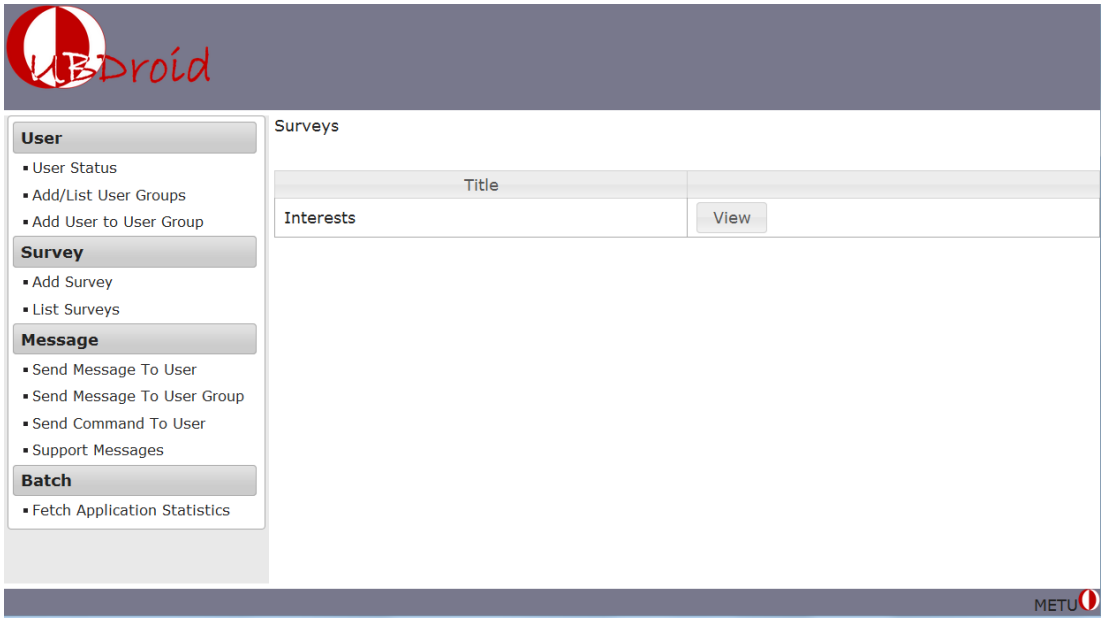

Figure 30 – List surveys web interface

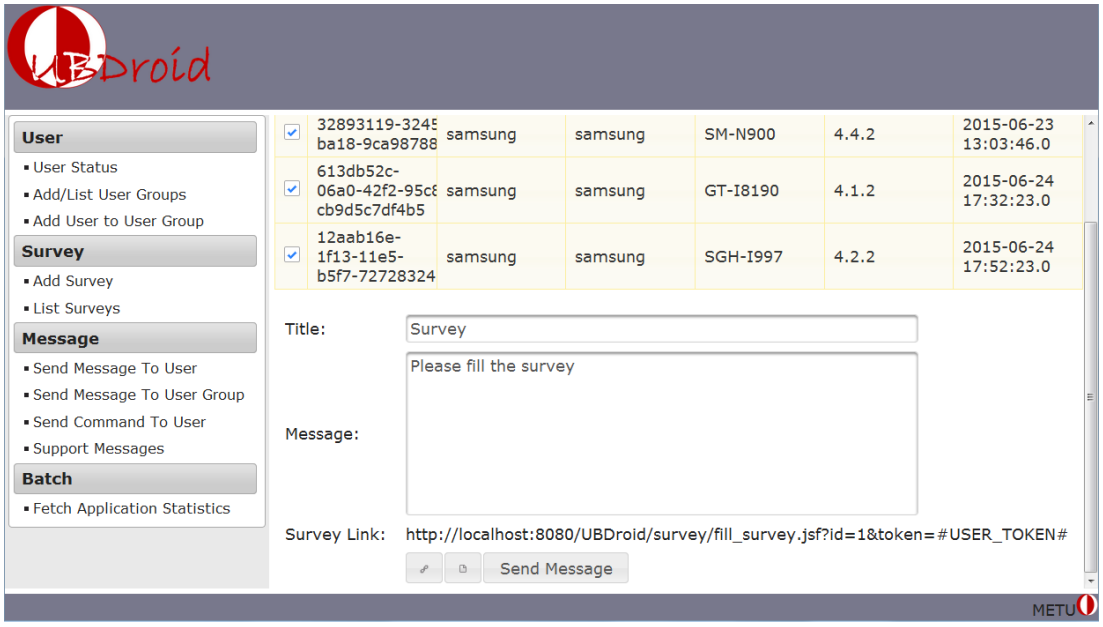

Figure 31 – Send message web interface

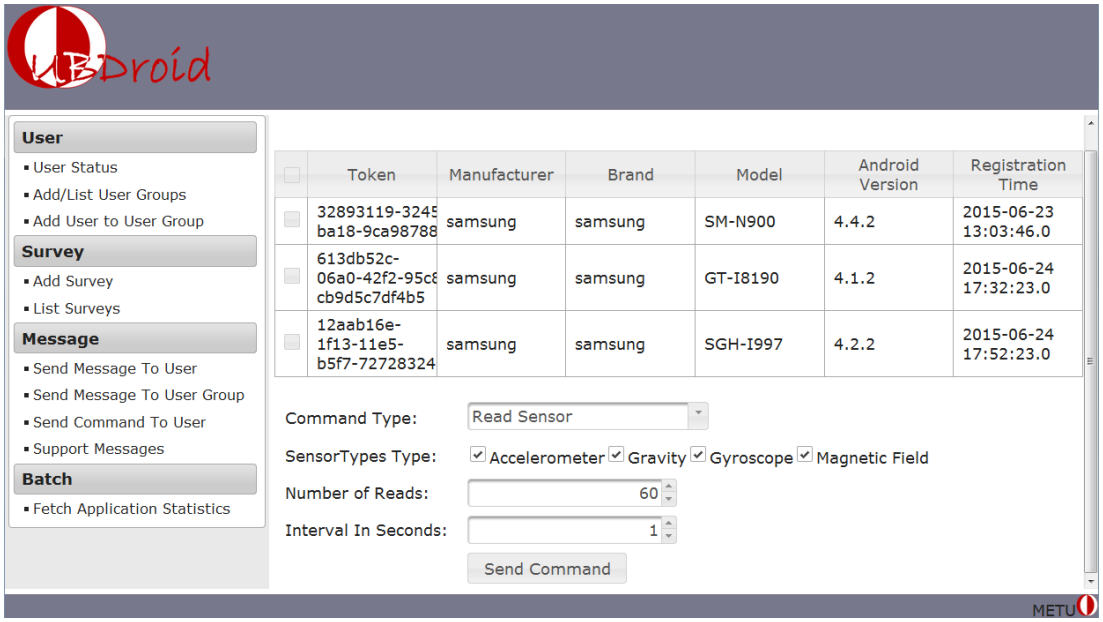

Figure 32 – Send command message web interface

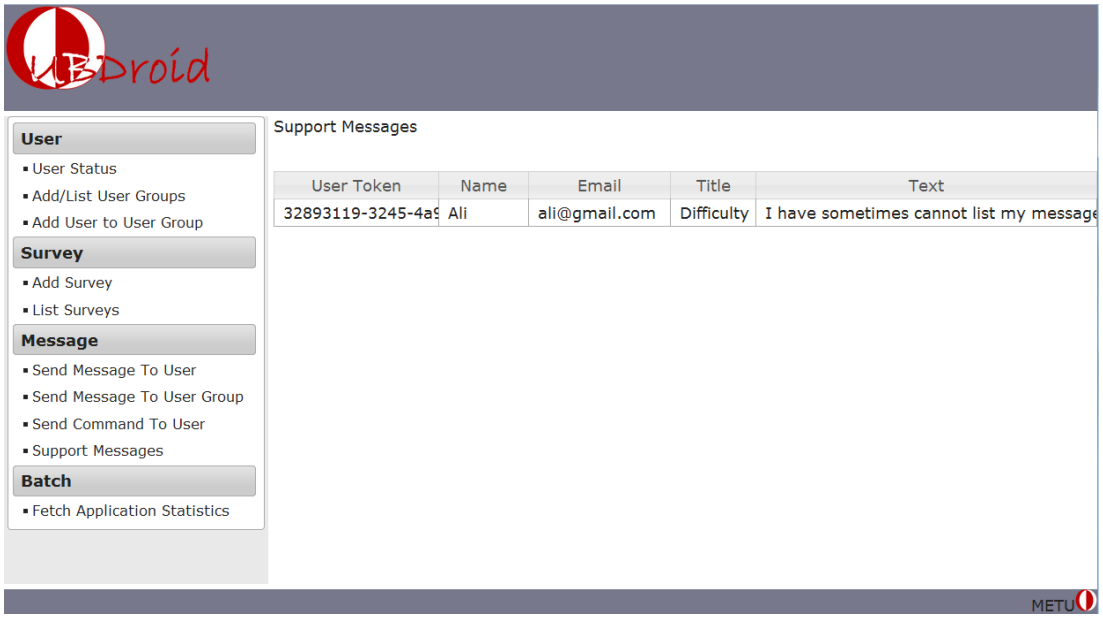

Figure 33 – Support messages web interface

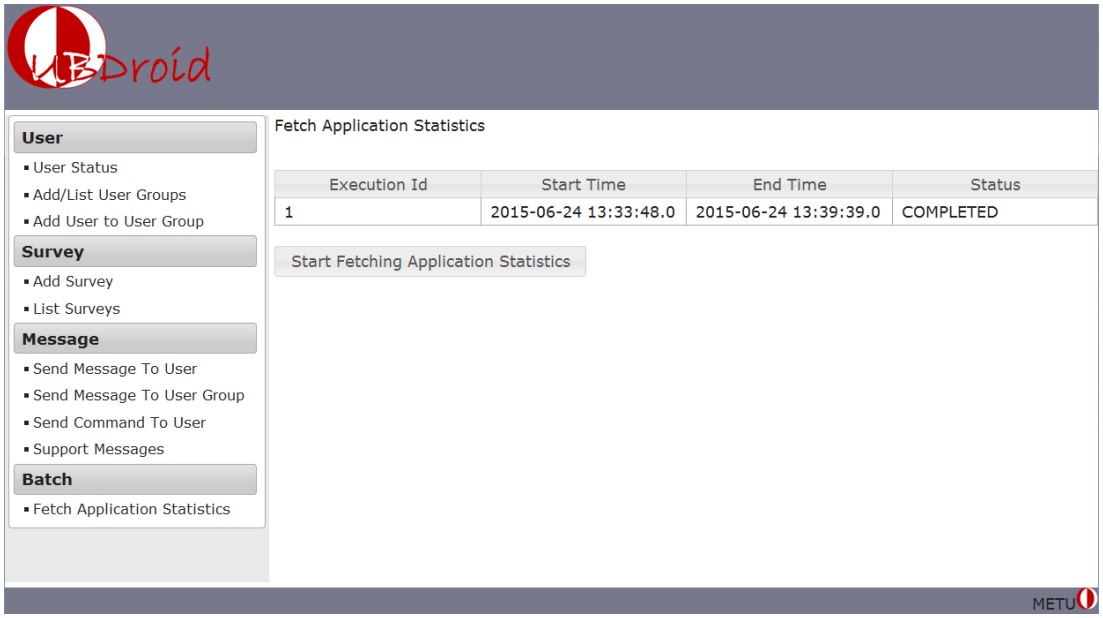

Figure 34 – Fetch application statistics web interface

Appendix B: Lessons Learned and Implementation Issues

1. org.hibernate.LazyInitializationException while returning an entity:

The problem occurs while converting Hibernate proxy objects to JSON format:

- Entities are fetched from database and mapped to Hibernate proxy objects.
- JSON converter traverses fields of returned entity.
- Calling lazy fields of the entity triggers database query. Since there is no database transaction while JSON conversion is performed, Hibernate throws exception.

There are two solutions for the problem:

- Always return DTO from server, so all the required fields of entity should be copied to DTO.
- Make a JSON conversion so that lazy fields should not be included.

The second solution leads lower development costs from the first one. We moved from Gson to Jackson library since Jackson has Hibernate aware conversion module.

2. Empty selection does not properly work for combo boxes at web client application:

Primefaces provides a mechanism for binding objects to ui components. In fact, there is no object at HTML front-end, unlike back-end which is run by Java. Primefaces solves the problem by providing string to object and object to string converter mechanism. Empty selection problem is caused by improper use of converters. For empty selection, object to string conversion should not return null, but return empty string.

3. List of entity loses type information while deserialization:

Java generic types cannot be determined at runtime, so Jackson converter needs type information to be defined explicitly. Using array type or putting list of entity in a DTO are simpler solutions which do not affect default conversion configuration.

4. Time serialization-deserialization:

Jackson uses UTC by default. If both serialization and deserialization is done by Jackson library, there is no need for additional configuration. If server or client uses another JSON conversion library, then time format or time zone configuration must be done properly.

5. JSF or Spring Framework annotations cannot be used at the same time:

When JSF and Spring are integrated, it is inevitable to face the problem. Programmers are mostly intended to use JSF's ViewScope (to manage view-controller interaction) and Spring's Autowired (to manage dependency injection) annotation in same controller. However, Spring annotations do not work, when JSF annotations are resolved. We overcome the problem by providing custom implementation for view scope, and use only Spring annotations.

6. EntityManager vs SessionFactory:

Both EntityManager and SessionFactory are intended to perform CRUD operation at data layer. EntityManager is part of JPA standard whereas SessionFactory is specific to Hibernate. Since Hibernate fulfils JPA standard, it is possible to use EntityManager with Hibernate. Best practice is to use EntityManager by default and accessing Hibernate specific features where needed. EntityManager allows getting Hibernate session by calling unwrap function; *EntityManager.unwrap(Session.class)*.

7. Creating REST services with Spring MVC:

For REST services, Spring MVC requires a servlet and a context configuration for the servlet to be defined. REST context is created as child of application context, so core application services can be accessed from REST services, but not vice versa. However, improper REST context configuration makes application services unavailable for REST services. To solve the problem, duplicating creation of application services in REST context configuration is a common mistake. Best

practice is to have only creation of communication layer services in REST context configuration.

8. Sensor read is stopped after 15 seconds on some devices when the device is in sleep mode:

We found that the problem occurs because of manufacturer settings. Partial wake lock is acquired to ensure sensor can be read in sleep mode.

REFERENCES

[1] "Wikipedia," [Online]. Available: http://en.wikipedia.org/wiki/List_of_mobile_software_distribution_platforms. [Accessed 10 01 2015].

[2] "Apple Press Info," [Online]. Available: http://www.apple.com/pr/library/2013/05/16Apples-App-Store-Marks-Historic-50-Billionth-Download.html. [Accessed 10 01 2015].

[3] "Apple Press Info," [Online]. Available: http://www.apple.com/pr/library/2014/01/07App-Store-Sales-Top-10-Billion-in-2013.html. [Accessed 10 01 2015].

[4] Business Insider, [Online]. Available: http://www.businessinsider.com/iphone-v-android-market-share-2014-5. [Accessed 10 01 2015].

[5] "IDC," [Online]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp. [Accessed 10 01 2015].

[6] "StrategyAnalytics," [Online]. Available: https://www.strategyanalytics.com/default.aspx?mod=pressreleaseviewer&a0=5505. [Accessed 10 01 2015].

[7] E. Kepucka, "A Mobile Sensing Framework for Audience Emotion Analysis," M. S. thesis, METU, Ankara, 2014.

[8] "Funf Open Sensing Framework," [Online]. Available: http://www.funf.org/. [Accessed 10 01 2015].

[9] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury and A. T. Campbell, "A Survey of Mobile Phone Sensing," *IEEE Communication Magazine,* 2010.

[10] H. Lu, J. Yang, Z. Liu, N. D. Lane, T. Choudhury and A. Campbell, "The Jigsaw Continuous Sensing Engine for Mobile Phone Applications," in *SenSys'10*, Zurich Switzerland, 2010.

[11] H. Verkasalo, "Analysis of Smartphone User Behavior," in *Ninth International Conference on Mobile Business*, Athens Greece, 2010.

[12] P.-M. Chen, C.-H. Chen, W.-H. Liao and T.-Y. Li, "A Service Platform for Logging and Analyzing Mobile User Behaviors," in *Proceeding of Edutainment*, 2011.

[13] R. LiKamWai, Y. Liu, N. D. Lane and L. Zhong, "MoodScope: Building a Mood

Sensor from Smartphone Usage Patterns," in *MobiSys'13*, Taipei Taiwan, 2013.

[14] V. Pejovic and M. Musolesi, "InterruptMe: Designing Intelligent Prompting Mechanisms for Pervasive Applications," in *UbiComp'14*, Washington USA, 2014.

[15] B. Yan and G. Chen, "AppJoy: Personalized Mobile Application Discovery," in *MobiSys'11*, Maryland USA, 2011.

[16] X. Wei, L. Gomez, I. Neamtiu and M. Faloutsos, "ProfileDroid: Multi-layer Profiling of Android Applications," in *MobiCom*, Istanbul Turkey, 2012.

[17] S. Lee, C. Yoon and H. Cha, "User Interaction-based Profiling System for Android," in *UbiComp*, Seattle USA, 2014.

[18] H. Falaki, R. Mahajan and D. Estrin, "SystemSens: A Tool for Monitoring Usage in Smartphone Research Deployments," in *MobiArch'11*, Maryland USA, 2011.

[19] "Phone Arena," [Online]. Available: http://www.phonearena.com/news/A-modern-smartphone-or-a-vintage-supercomputer-which-is-more-powerful_id57149. [Accessed 10 01 2015].

[20] "Search Engine Watch," [Online]. Available: http://searchenginewatch.com/sew/opinion/2353616/mobile-now-exceeds-pc-the-biggest-shift-since-the-internet-began. [Accessed 15 01 2015].

[21] "Sensors Overview," [Online]. Available: http://developer.android.com/guide/topics/sensors/sensors_overview.html. [Accessed 10 01 2015].

[22] N. D. Lane, S. B. Eisenman, M. Musolesi, E. Miluzzo and A. T. Campbell, "Urban Sensing Systems: Opportunistic or Participatory," in *HotMobile*, NY USA, 2008.

[23] X. Ma, P. Huang, X. Jin, P. Wang, S. Park, D. Shen, Y. Zhou, L. K. Saul and G. M. Voelker, "eDoctor: Automatically Diagnosing Abnormal Battery Drain," in *10th USENIX Symposium on NSDI*, Lombard IL, 2013.

[24] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller. [Accessed 25 07 2015].

[25] "Spring Framework," [Online]. Available: http://docs.spring.io/spring-batch/trunk/reference/html/domain.html. [Accessed 10 01 2015].

[26] F. Cristian, "Probabilistic Clock Synchronization," *Distributed Computing*, vol. 3, no. 3, pp. 146-158, 1989.

[27] "Android Developers," [Online]. Available: http://developer.android.com/tools/debugging/ddms.html. [Accessed 21 06 2015].

[28] "Google Play," [Online]. Available: https://play.google.com/store/apps/details?id=edu.umich.PowerTutor. [Accessed 21 06 2015].

[29] "Google Play," [Online]. Available: https://play.google.com/store/apps/details?id=ccc71.at. [Accessed 21 06 2015].

[30] "Apache JMeter," Software Foundation, [Online]. Available: http://jmeter.apache.org/. [Accessed 21 06 2015].

[31] "Android Developers," [Online]. Available: http://developer.android.com/reference/android/app/ActivityManager.html. [Accessed 21 06 2015].

TEZ FOTOKOPİSİ İZİN FORMU

ENSTİTÜ

Fen Bilimleri Enstitüsü ☐

Sosyal Bilimler Enstitüsü ☐

Uygulamalı Matematik Enstitüsü ☐

Enformatik Enstitüsü ☒

Deniz Bilimleri Enstitüsü ☐

YAZARIN

Soyadı : Akkurt

Adı : Erkam

Bölümü : BİLİŞİM SİSTEMLERİ

TEZİN ADI (İngilizce) : UBDROID: A TOOL FOR MONITORING SMARTPHONE APPLICATION USAGE FOR USER BEHAVIOR ANALYSIS

TEZİN TÜRÜ : Yüksek Lisans ☒ Doktora ☐

1. Tezimin tamamından kaynak gösterilmek şartıyla fotokopi alınabilir. ☒
2. Tezimin içindekiler sayfası, özet, indeks sayfalarından ve/veya bir bölümünden kaynak gösterilmek şartıyla fotokopi alınabilir. ☐
3. Tezimden bir (1) yıl süreyle fotokopi alınamaz. ☐

TEZİN KÜTÜPHANEYE TESLİM TARİHİ : …….