INTERNATIONAL
HELLENIC
UNIVERSITY

Context-Aware Profile Analyzer for Android

Kotsopoulos Dimitrios

SID: 12345678

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Information and Communication Systems

DECEMBER 2015

THESSALONIKI – GREECE

INTERNATIONAL HELLENIC UNIVERSITY

Context-Aware Profile Analyzer for Android

Kotsopoulos Dimitrios

SID: 12345678

Supervisor:	Prof. T.-M. Groenli
Supervising Committee Members:	Prof. G. Ghinea
	Dr. C. Berberidis

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Information and Communication Systems

DECEMBER 2015

THESSALONIKI – GREECE

Abstract

In this thesis we endeavor to develop a prototype of a mobile application that focuses on context-aware computing, trying to deploy it in a new and novel way and investigate unexplored dimensions of context-awareness. As users receive calls and texts, browse the web, play games and use applications, information aggregates in the phone. Our scope is to try to extract this information from communication log files of mobile device as well as from device's sensors. Incoming and outgoing calls, messages or emails from several installed applications on mobile phone can be considered really useful for profile analysis of mobile owner. By combining this information with traditional context-aware information from sensors and build in data stores in the phone, an extensive user profile can be built an analyzed. Furthermore, we try to enlighten the reader regarding the concept of context and context-awareness in order to provide her a general knowledge for various frameworks which application developers used in order to develop context-aware application and finally we present several prototype applications. Furthermore, we focus on defining the problem together with the approach and the methodology which will be used. In addition, we present the tools that we used for the completion of this thesis together with the core implementation of the prototype application. The conclusion shows that the given approach is feasible to create a context-aware application based on simple design methodology using as essential information data from mobile sensors and user's communication log.

Kotsopoulos Dimitrios

11/12/2015

Contents

1 Introduction

1.1 Background

Never before in modern history of humanity, was man so closely intertwined with her personal device (desktop pc, laptop, smartphone etc.) as now. The first years of development of computer technology, a user had in her possession a desktop computer with specific capabilities, and then computers have evolved rapidly over years and became portable with more capabilities. This development did not leave unaffected the users who in turn begun to possess more than one computer and especially at least one laptop. The dependency created between devices and users was great and became even stronger when the smartphones were appeared. Mobile experiences overtook the desktop experience. In 2013, Gartner Group [1] predicted that mobile devices will pass PCs to be most common Web access tools. They also predicted that by the end of 2015, over 80% of handsets in mature markets will be smart phones. IDC [2] predicted that smartphone and tablet spending will reach $484 billion; generating 40% of all IT growth whereas mobile app downloads will hit 150 billion, up 18% from last year's 61% growth. In 2015, over 3.5 million applications are available across app stores worldwide. Based on the previous surveys it is safe to assume that in the developed world almost every individual holds at least one smart device which uses some kind of smart automation that perhaps the user is not aware.

How many times have we used our mobile in vertical or horizontal display and device's screen followed our motions? How many times have drivers used a navigation application to plan a trip or get information about a route? We can enumerate several similar examples that the user takes for granted such the above functionalities for mobile applications that facilitate her daily life. Mark Weiser in his pioneering paper [3] quoted that *the most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it*. To make this possible, many researchers from the 90s tried to find out what are the necessary ingredients, which can be used to create such systems, context-aware systems. Universal importance step is the effective exploitation of contextual information. Interaction between computers and humans has many differences in relation with human to human, computers understand only what they are programmed to understand whereas humans can understand the environment, to distinguish facial expression, change the tone of voice of the interlocutor even know news related to speakers. In the current decade the development of many sensors enabled the enhancement of context. Nowadays in most mobile devices and not only (objects etc.) are embedded important sensors such as GPS (for location and speed), light, vision, microphones, accelerometers, gyroscopes, magnetic field sensors, sensors for temperature and humidity, and air /barometric pressure. Additionally many wireless computer networking technologies have been improved and used to provide secondary context, Wi-Fi, Bluetooth, GPRS, infrared, NFC etc. The following figure represents with accuracy what was mentioned before and in addition shows the trend of technology for years to come, namely Internet of Things. Guillemin and P. Friess in their research work [4] defined that the *Internet of Things allows people and things to be connected anytime, anyplace, with anything and anyone, ideally using any path/network and any service*. Objects around us will connect to each other (e.g. machine to machine) communicate via the Internet and sharing context. **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.**illustrates the phases in the evolution of the Internet.

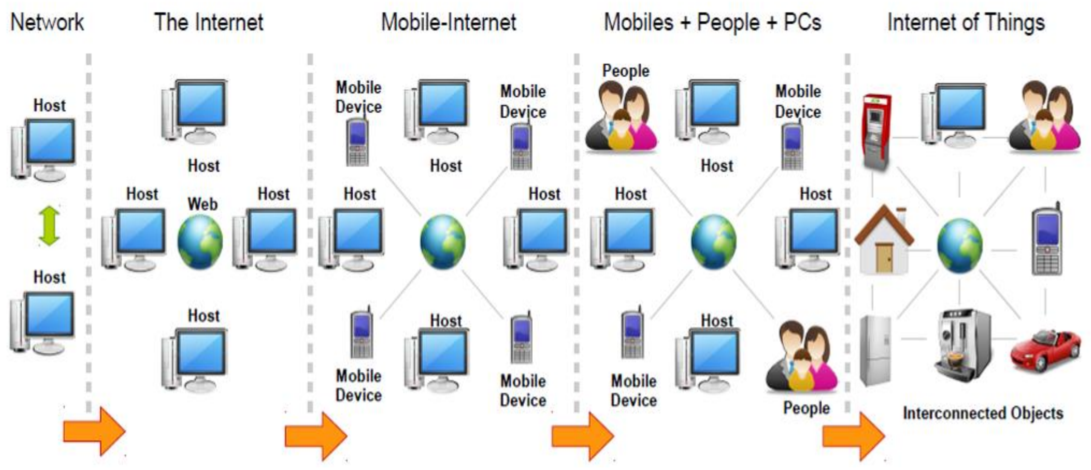


Figure 1 Evolution of Internet [5]

Several companies in the IT sector such as Google, Yahoo and Facebook push towards to this direction and they significantly influence the future of Mobile. Google for example has already demonstrated a context-aware scenario where the smart phone assists her owner. A smart phone would wake up the owner based on the time that she slept but also on the current traffic, in order to arrive on time to her work, that it has been informed by web services. Then it will open up the lights and also increase the temperature in the room due to the fact it is connected to light and temperature sensors. It will turn the coffee machine on and once the user is on the computer it will automatically present to him news of her preferences. When the owner would get into the car, lights and temperature in the house will be automatically turned off, and the smartphone will present the optimal route for her work and it will turn on the radio station of driver's choice. The scenario can continue even more and provide to the user an indicator regarding the parking spots or where he has already parked her car etc. This is only a fascinating part of what the future may hold for context-aware computing.

1.2 Thesis Relation

In this thesis we will endeavor to develop a mobile application that focuses on context-aware computing and we will seek to deploy it in a new and novel way and to explore unexplored dimensions of context-awareness. We will try to extract data from communication log files of mobile as well as data from device's sensors. Incoming and outgoing calls, messages or emails from several installed applications on mobile phone can be considered really useful for profile analysis of mobile owner. Additionally, we will try to integrate specific services in order to provide user additional aim in specific occasions, namely while waiting one bus stops to display the timetable or at the end of work to display the activities in that area such as cinema or various other music events based on the preferences of user or user's location. Finally, we will try to create a profile on the basis of the application that user has on her mobile and optimize her phone performance (battery consumption, profile switching etc.)

1.3 Goals and Expected Outcomes

The goal of this dissertation is to develop an android mobile application for gathering of context-aware data. More specifically within the installed applications we will seek to:

- Develop the architecture to be used for the implementation of the context-awareness prototype application;
- Pull the communication logs so that we provide a statistical representation to user and also exploit them for further analysis for an extensive user profile;
- Provide appropriate phone profile based on Google calendar or other sources (If user has a meeting the phone will switch to mute mode);

- Provide information regarding events based on sensor information, GPS, gyroscope, accelerometer, etc.;
- Provide information regarding the public transport based on user’s location.

1.4 Thesis Outline

In Section 2 *Literature Review* we will define what is context and context-awareness in order to provide the reader a general knowledge for the upcoming literature review. We will also see various frameworks which application developers used in order to develop context-aware application and finally we will present few prototype applications.

In Section 3 ***Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.*** we focus on defining the problem together with the approach and the methodology which will be used. In addition we present all the tools to be used for the completion of this thesis.

In Section 4 ***Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.*** we will present the core implementation of the system and we will endeavor to achieve a combination of context-awareness data with phone data aimed at a creation of a profile analyzer.

In Section 5 *Discussion* we will discuss our results as described in previous sectors with our informed background reading our stated research problem.

In Section 6 *Conclusion and Future Work* we will restate our position, what was learned from the dissertation and we will complete the thesis with the future work that remains to be learnt based on what we have researched in the present thesis.

2 Literature Review

In this section we will define the meaning of context and context-awareness in order to provide the reader a general knowledge for the upcoming literature review and the related work. Additionally we will try to summarize, without diminishing the substance of the writer's labor, a set of frameworks and applications introduced during the scientific efforts throughout the years.

2.1 What is context

Communication of people, beyond the speech, is achieved to a large extent with the aim of body movements, expressions of person, the background of interlocutors and facts or situations that take place at the moment of people's communication. The dictionary of Merriam-Webster defines context as *the surroundings, circumstances, environment, back-ground or settings that determine, specify, or clarify the meaning of an event or other occurrence*. A big question that deplores years the researchers of computer science lies in the fact that computers can comprehend only raw data that is inserted as input. This information is very difficult to encapsulate automatically emotions, situations and moves and be offered as entry to computer systems. From the decade of the 90's there have been attempts by scientists to embody a range of information from sensors into innovative applications in order to provide appropriate services in the appropriate users, at the right time and in the right place. But what is the context in mobile computing or ubiquitous computing? In 1994 Schilit and Theimer in their project [6] referred to the term context as the location, people and objects nearby as well as changes made to the formers over time. Brown et al [7] defined context as user's location, time of day, season of year, temperature etc. We can continue enumerate a plurality of researchers (Franklin, Flaschbart, Hull, Pascoe etc.) and their definition, however almost all definitions are synonymous or identical and do not escape from the broad definition given by Dey in his PhD thesis [8] where he defined context as: *Any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves*.

Schilit, Adams and Want [1] endeavored to specify context to three categories:

- Computing context: For example, network connectivity, communication band-width, nearby resources like printers, displays;
- User context: For example, user's profile, location, emotional state, people nearby, current activity;
- Physical context: For example, lighting, noise level, traffic conditions, temperature.

However primary context (location (where), identity (who), time (when), and activity (what)) is more important than others as Dey [8] noted.

2.2 Context-awareness

Context-awareness in essence refers to the pursuit, the possibility of computer systems to perceive and react based on the information received from the surroundings. Schilit and Theimer [9] consider that context-aware software should adapt according to the location, nearby people, objects, as well as to changes to such things over time. Dey [8] highlighted that systems such as context-aware system use context to provide the relevant information to the user based on her preferences or her current state. Studies regarding the features of context-aware have been implemented and the three most prevalent are those of Schilit, Adams and Want [6], Pascoe [10] and Abowd et al. [11] where the first two efforts provided two different groups of features with similarities which Abowd et al. [11] exploited and created three features uniting commonalities of foregoing researchers. We list below the features of the three research groups. The first list is from Schilit, Adams and Want who provided the following taxonomy:

- Proximate Selection: A technique that selects nearest object that is related to the location of the user. Displaying a list of colleagues located around the user can be considered proximate selection;
- Automatic Contextual Reconfiguration: This process is responsible for varying the components of the application or the system. This change, insertion, deletion or any other is done automatically by the application environment. As in our previous example, the list of colleagues can be changed, either by coming more colleagues at the spot or to leaving the spot;
- Certain information and commands are being revealed to the user based on her current location. The information change together with the commands that can be exe-

cuted by the system. When the user is on the corridor, system can display information about the temperature of the area or their colleagues who are in office; when the user is located in the kitchen the system can display information about the coffee temperature;

- Context triggered actions: Context triggered actions are just simple IF-THEN rules used by the system in order to adjust to the environment. These rules provide the way of adapting to possible changes. One example is the entrance of a particular user in kitchen at a particular hour will trigger a specific event.

In the second list Pascoe 10] presented his different approach:

- Contextual Sensing: The process of detecting environmental states and illustrating to the user defines contextual sensing. As Pascoe wrote a GPS application can detect the latitude and longitude and present this location to the user via map visualization;
- Contextual Adaption: When context is available to the system, the system itself can modify its processes and adapt automatically its behavior to environmental states. Instead of presenting to the user the contextual data the system can customize its context. Interchangeable orientation can be an application's feature which based on the direction the user interacts; the display can change to horizontal or vertical view;
- Contextual resource discovery: Based on the relevancy of user's or other entity's context the system can discover and exploit resources and services. In other words contextual resource discovery takes account not only its own context but also context from entities which do not belong in its limited environment;
- Contextual augmentation: Contextual augmentation is the ability to augment additional information to system's environment thus combining relative digital data and the user's context. A digital tour guide is a popular application that applies contextual augmentation. Museums use annotation to each and every painting or statue and when a user is in front of a specific object comments are being displayed to user's screen regarding the concerned object.

Finally, Abowd et al. [11] proposed a combined categorization based on the aforementioned approaches:

- Presentation: It is a combination of Schilit's [6] proximate selection and contextual commands and also Pascoe's contextual sensing. The presented information or services are determined by context. A user can see an updated list of nearby shops as she moves inside a shopping center;

- Execution: Execution is referred to the automatic execution of services which is a critical feature for context-aware applications. It similar to Schilit's [6] context-triggered actions and Pascoe's contextual adaption. A sophisticated example of automatic execution is when a fridge understands the lack of milk and communicates automatically with the supermarket to order milk supplies. This requires machine to machine communication, nowadays a widely expanding field named Internet of Things;
- Tagging: It is the same feature as Pascoe's [10] contextual augmentation. System uses tags for subsequent retrieval. User can use stick notes to inform other users that a television in a specific room is malfunctioning. When a new user enters the room, she will be automatically notified by the system regarding television's state.

2.3 Related work

There have been many attempts in the course of years by research centers and companies to take advantage of the mobile and ubiquitous/pervasive computing, some flourish and created context-aware applications others established frameworks and prototype applications. Below are shown some important implementations which helped later development of context-aware applications, (F) is referred to frameworks and (A) to applications

2.3.1 Context Toolkit (F)

Dey in his PhD thesis [8] suggested a design process based on three important procedures, specification of the problem and the needed context, acquisition of the appropriate equipment (hardware, sensors etc.) and finally action for choosing and performing context-aware behavior. He also argued that a distinction between context and user input is necessary due to the fact that in context-aware applications, context is derived from many computers-sources and also context required additional process in order to be valuable to applications. His research provided a framework, influenced by Graphical User Interfaces, which helped application developers to develop superior context-aware application liberated from the shackles of sensors. The main abstractions were BaseObject, Widget, Aggregator, Interpreter and Discoverer:

- BaseObject: Provides the appropriate channels so as to communicate effectively the remaining components;
- Widget: Is responsible for retrieving data from the sensors and providing homogeneous interface for all the components or applications that intent to use the data. Inside Widget there is a secondary component *services* that are responsible for the actions of retrieving or providing data as outcome;
- Aggregator: Is similar to Widget, the only difference being is that aggregators can collect all the context regarding an entity. The context is retrieved by the Widgets and it is a mediator between the Widgets and the rest of the components;
- Discoverer: It can be likened to "Yellow Pages" of the system. It helps the application or other components to find the appropriate context component. It automatically updates the list of available components by adding new ones or deleting old ones;
- Interpreter: It can take the low-level information from widgets and combine context from various sources (location, voice, etc.) through different reasoning techniques interpreting them to high level information for example longitude and latitude to street name.

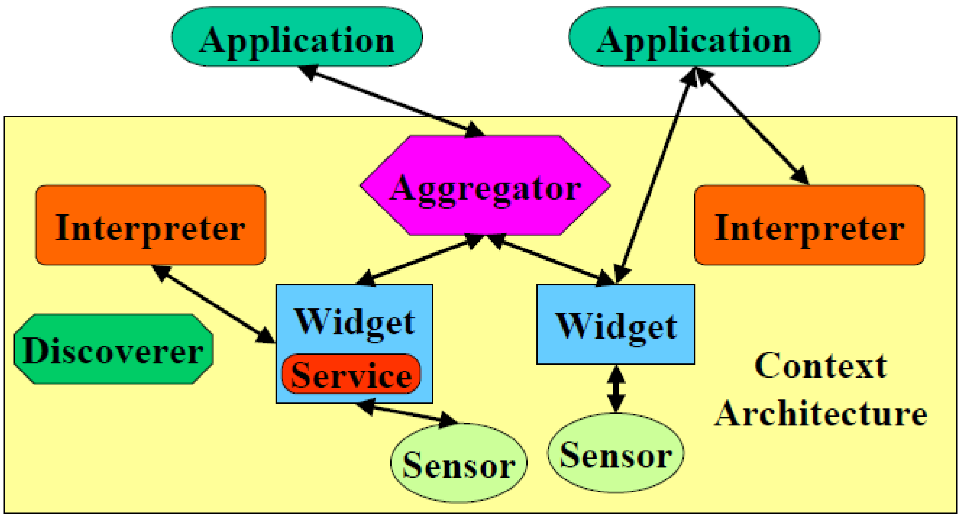


Figure 2 Context Toolkit components [8]

Various applications (In/Out Board, DUMMBO, etc.) were developed based on the Context Toolkit in order to prove that this framework can aim developers to design new context-applications.

2.3.2 CoBrA (F)

Context Broker Architecture [12] is a middleware agent architecture which supports knowledge sharing and context reasoning for smart spaces. CoBrA uses Semantic Web Languages both for context knowledge representation and reasoning namely Web Ontology Language (OWL) and RDF (Resource Description Format. The central pillar of this architecture is Context brokers and have several responsibilities such as providing a centralized model of context, reasoning about low-level context and privacy issues inside the smart place where operate. A context broker consists of four basic components: context knowledge base (knowledge is represented in RDF triples and stored database), context reasoning engine (performs reasoning in stored data via OWL's semantics), context acquisition module (fetches contextual information from available sources) and policy management module (provides access/denial priorities to agents based on the policies). An application that follows this architecture is EasyMeeting System [12] an intelligent meeting room system, which provides assistant to the participants based on the contextual information.

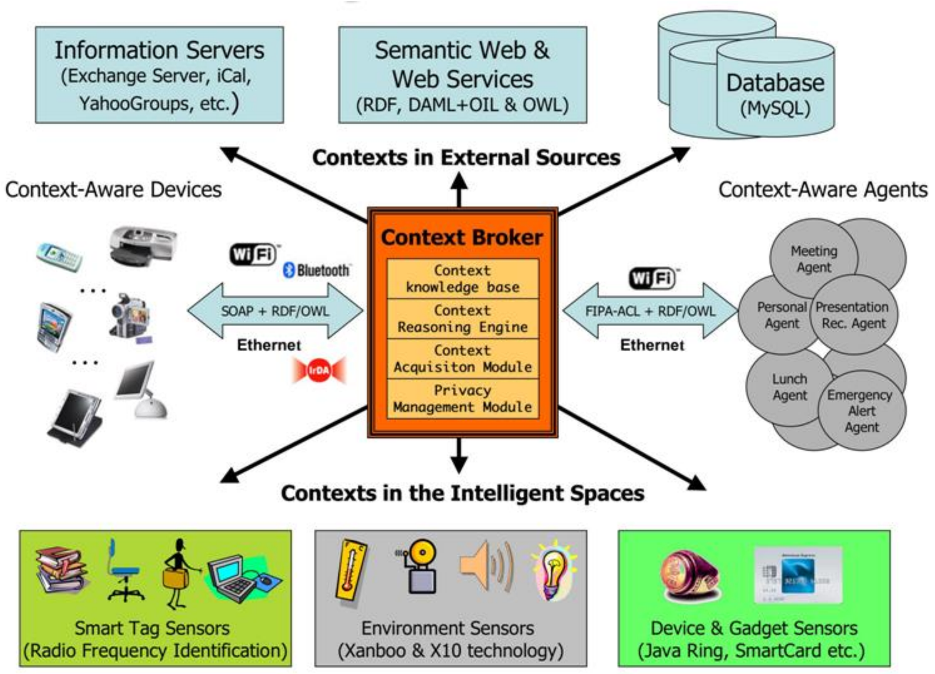


Figure 3 CoBrA Architecture [12]

2.3.3 SOCAM (F)

T. Gu, H. K. Pung, and D. Q. Zhang in their research work [13] proposed an architecture for the building and rapid prototyping of context-aware services namely Service-Oriented Context-Aware Middleware (SOCAM). It is based on OWL [14] to represent, manipulate and access context information. OWL context modeling can provide the proper mechanism for context reasoning where high level context can be derived from ,SOCAM middleware the context is represented as a triplet which has the form of Predicate (subject, value), in which subject is a set of subject names, predicate is a set of predicate names and value is a set of all values of subjects. A simple example of such a triplet is Eats (Alex, Apples) which imply that Alex eats apples. The research team separated the ontologies into two levels: upper level ontology for general concepts and lower level ontologies domain specific descriptions. SOCAM architecture is consisted of five key components as figure below illustrates:

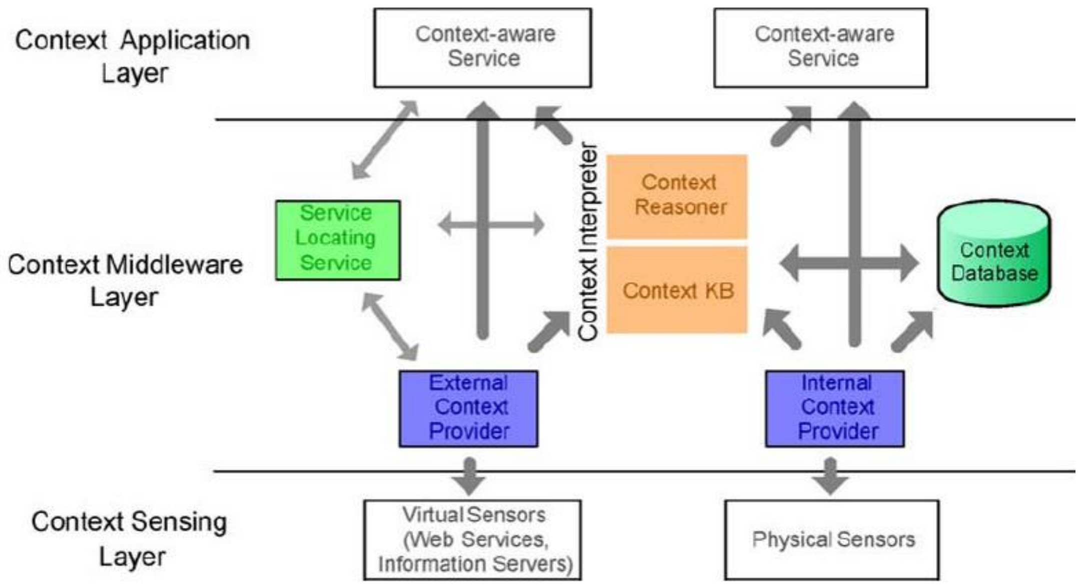


Figure 4 SOCAM Architecture [13]

- Context provider. It acquires data from sensors and other internal and external data sources and converts the context in to OWL vocabulary;
- Context interpreter. It provides logic reasoning services (Context Reasoner) and stored the outcome in the database (KB);
- Context database. It stores context ontologies and past contexts for a sub-domain. There is one logic context database in each domain, i.e. home domain;
- Context-aware services. They are the consumers of the system. They can use the context or adapt accordingly by creating action triggers whenever context is changing;

- Service locating service. Context providers and Context interpreter registered to the system and other components can seek for the appropriate provider or interpreter based on their needs.

2.3.4 CARISMA (F)

L. Capra, W. Emmerich, and C. Mascolo, in their research work developed CARISMA [15] a Context-Aware Reflective middleware System for Mobile Applications which focused on extremely dynamic systems such as mobile systems. CARISMA uses the reflection (Adaption) paradigm to enhance the adaptive and context-aware mobile applications development. As we know mobile devices change their context at a glance, however, context is monitored by the middleware and such context configurations determine the applied policies for a service affecting the customization of middleware behavior which dynamically is altered by the contextual information of the applications. CARISMA exploit the XML encoding in order to store application profiles which are the association of customized services by middleware, policies for invoking the services and context configurations to use the policies for example, if battery is below 40% use service A else service B. There are occasions when reflection can introduce conflicts; such cases are when more than one policy is valid. These conflicts could be Intra-profile conflict (local to middleware instance i.e. on the same device) or Inter-profile conflict (Various instances i.e. different devices). To encounter this issue, researchers created a mechanism based on microeconomic techniques. The platform uses an auction protocol which consists of a set of rules that meet specific requirements on the solution, dynamicity, simplicity and customization. The mobile system can be referred as economy where the consumers (applications) reach an agreement about a limited set of goods (a policy-battery level, CPU or available memory) using the middleware platform like auctioneer. The final decisions are made in order to maximize the social welfare among the applications.

2.3.5 Gaia (F)

Gaia [16] is a distributed middleware that provides similar functionality to an operating system. It is a middleware infrastructure capable of managing resources contained in physical spaces (active spaces). Active Space is defined as a *physical space coordinated by a responsive context-based software infrastructure that enhances the ability of mobile users to interact and configure their physical and digital environment seamlessly.*

The architecture of Gaia consists of three main building blocks as the figure below illustrates: the Active space applications, the Application framework and the Gaia kernel.
The Active space application contains registered applications and provides managerial functionalities (update, delete, register, and control) for the manipulation of applications through the Gaia Kernel services. The Application framework is a variation of the Model-View-Controller [17] and is composed of four elements: model, controller, presentation and coordinator. Coordinator is responsible for managing the application architecture. Finally Gaia kernel consists of two basic components: Component management core and services. Component management core is responsible for uploading, destroying, creating all the components and applications of Gaia. The Gaia kernel services are: Space repository (repository for software and hardware entities in the active space), event manager (distribute events and provide communication), context file system (data organization and transformation), presence (detect and maintain digital/physical entities) context (query and register contextual information aimed at adjusting to the environment).

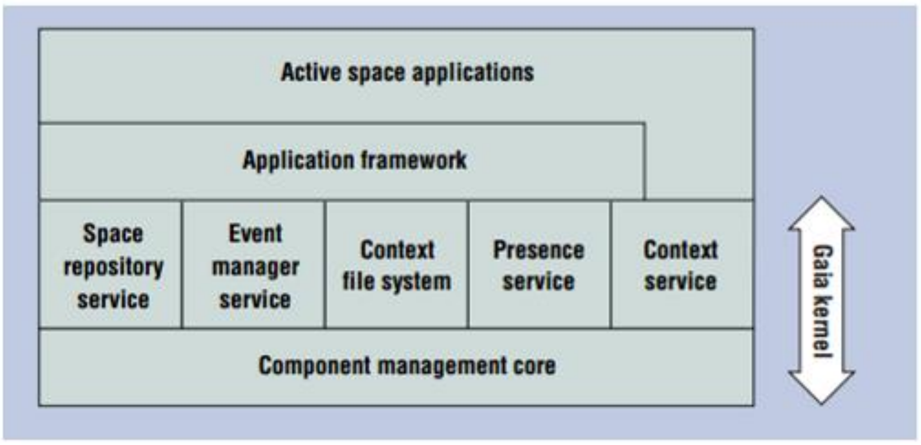


Figure 5 Gaia Architecture [16]

2.3.6 Aura (F)

Aura [18] is a task oriented system for distributed environments which users can use without concerns regarding the identity of device, environment or time. It runs on top of desktop operating systems in order to assist user to manage her tasks in all devices based on the contextual information. Moving from one point, where Wi-Fi signal is week, to another, where the signal is strong, without interruptions or the preservation of work during changes on work environments are the basic concepts of Aura. Aura does not support building application but offers its management services as intermediate of existing applications. In order to manage the services, application or the context Aura introduced the *Task Layer (Prism)* which captures user’s intentions. As the figure below illustrates, the key components of Prism are Task Manager which manages the transitions of tasks, Context Observer which collects context and notifies the Task Manager and Environment Manager for changes, Environment Manager manages the context supplier and specific services and lastly Service Supplier which implements the services needed to support a user’s task.

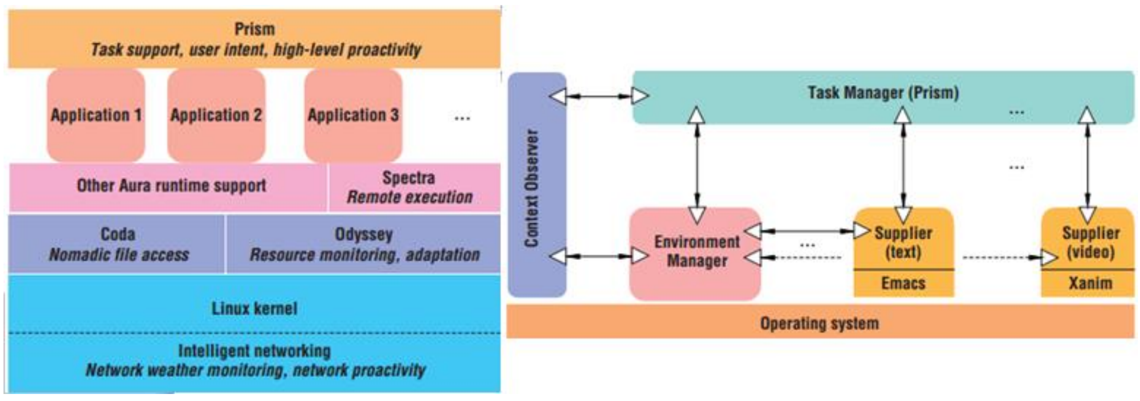


Figure 6 Aura Architecture and Prism Architecture [18]

2.3.7 COPAL(F)

COPAL [19] an adaptive approach to context provisioning is a middleware which provides loose-coupling between context and its processing. COPAL tried to provide not only a middleware framework but also a new design for developing context provisioning schemas. COPAL architecture consists of listeners, publishers and the core COPAL components as figure below illustrates.

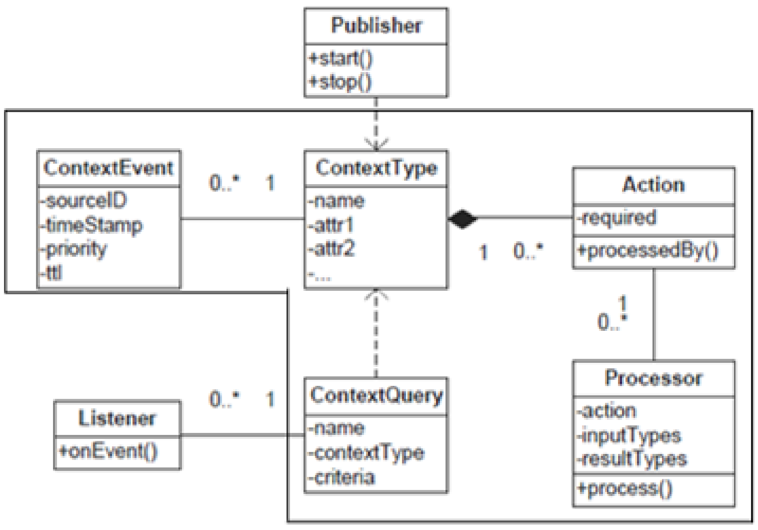


Figure 7 COPAL components [19]

Publishers: Publishers are all the available devices or sensors that communicate with the COPAL core components. Due to the large variety of hardware and protocols, publishers are wrapped as web services before they are connected to COPAL. In fact, Wrappers are those who communicate with the core and they may hold more than one publisher. Additionally, to this level a device manager exist which is responsible for storing information, maintaining the device/sensor catalog and monitoring the device/sensor status.

Listeners: Listeners are essentially the receivers of the notification that the COPAL provides when the appropriate context queries are fulfilled.

COPAL Core: The core consists of: a) the *context type* which has a unique name and a set of attributes, each publisher can publish a certain type of context, b) the *context query* which is responsible for the selection of the events, and again a query has unique name and a set of criteria, c) the *context event* which has specific attributes required or optional, d) the *context processor* which handles the action that it has been designated to perform, and e) action which are responsible of the insertion, modification and removal of context event’s attributes

In COPAL the context is delivered by events and its events has its own attributes, required or optimal, such as an identity of the source, a time stamp, priorities, the location of the source, the quality of the context etc. This context can be processes via filtering, abstracting, differentationing, enrichment or peeling in order to feed the outer events of COPAL.

2.3.8 CA4IoT (F)

CA4IOT [20] is a sensing-as-a-service middleware which tries to solve the single issue of selecting the most appropriate sensors according to the requirements of a single user. As the authors pointed out, this middleware is an assisting tool which should collaborate with other middlewares. The middleware uses XML language to represent context data related to sensors and also user can submit their requests for querying context using XML data format. The architecture is based on specific functional requirements such as the ability to connect sensors, to understand the contextual information of them, understand the user’s request, fill the gap between high level requirements and low level capabilities and mine high level context from low level raw data. An overview of CA4IOT architecture is displayed in figure below.

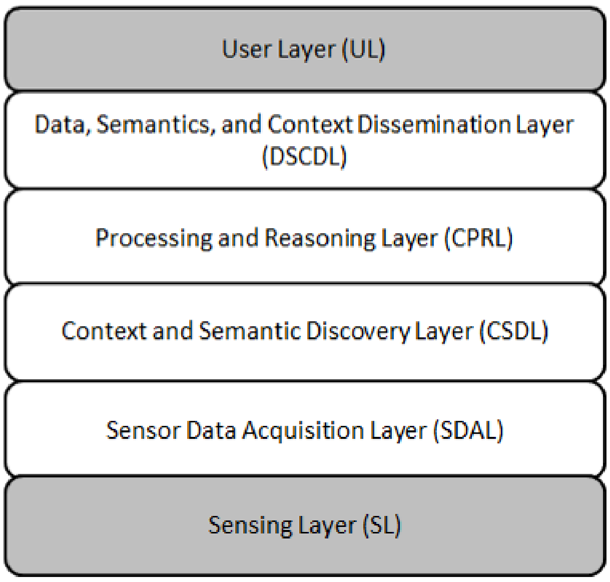

User Layer (UL)
Data, Semantics, and Context Dissemination Layer (DSCDL)
Processing and Reasoning Layer (CPRL)
Context and Semantic Discovery Layer (CSDL)
Sensor Data Acquisition Layer (SDAL)
Sensing Layer (SL)

Figure 8 CA4IOT architecture [20]

Four major layers compose the middleware which are listed as follows:

- *DSCDL*. This layer is responsible for the user's requests, data dispatching, and subscriptions and in general for user management. A user can make a request, and the system will interact with repositories (local or internet cloud) to provide an answer to user's request.
- *CPRL*. The most important component of CA4IOT where data processing, context reasoning, context fusing, knowledge generating and storing are occurred by specific functions.
- *CSDL*. This layer is responsible for context managing and semantic discovering. Relevant components are context and semantic discoverers, context and semantic discoverer generator, and context and semantic discoverers repository.
- *SDAL*. This layer is responsible for communication between the system and the sensors. It has modules such as sensor wrappers, wrapper repository, wrapper generator, local repository and SDD cloud repository. These modules acquire a variety of context data and retrieve them into CA4IOT.

It is worth mentioning that User Layer (humans, machines etc.) and Sensing Layer (software, hardware etc.) even though that are mandatory for a successful interaction they are not part of the middleware.

2.3.9 SeCoMan (F)

SeCoMan [21], which stands for Semantic Web-based Context Management, is a solution for developing context-aware smart applications focusing on preserving the user's privacy. In SeCoMan, Web Semantics are employed to model the description of entities, reason over data to retrieve useful contextual information, and define context-aware policies. There are three kinds of actors, namely Framework administrator who are responsible for managing the context resources and registering the smart applications; the Application Administrators who are responsible for managing the context of her own applications, handling the Context-aware ontologies, policies and queries; and finally Users who use the applications. The whole architecture of SeCoMan is shown in figure below.

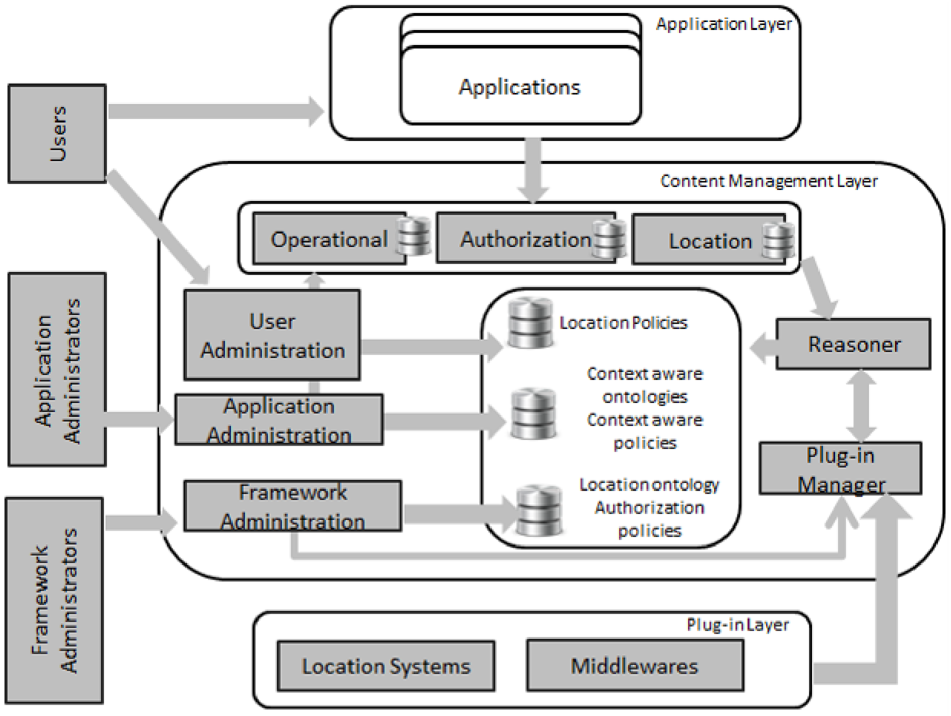


Figure 9 SeCoMan Architecture [21]

The SeCoMan architecture consists of three main layers namely Application, Context Management, and Plug-in (from top to bottom).

• Application. The application which reside on this layers offers specific location services and also make the proper queries to the Context Management in order to retrieve information desired by users;

• Context Management. As the core of the SeCoMan framework, it provides context-aware supports for applications. Three kinds of actors with different rights to interact with SeCoMan are defined including Framework Administrator, Application Administrator, and Users. A set of predefined queries are allowed for applications to get information about indoor location of users and objects. Semantic rules are used to specify policies regarding restricted access to location information so that privacy is guaranteed.

• Plug-in. It provides SeCoMan with contextual information and the space, which is especially focused on elements that form a part of the environment. In other words, the plug-in layer acts as an independent context source and obtains information, related to context from Middlewares (which communicate with sensors) and information from Location Systems related to location.

2.3.10 Olivetti Active Badge (A)

Researchers at Olivetti Research Ltd. (ORL) created the first context-aware application named Active Badge [22]. Users of this application wore Active Badges which transmitted a signal every 15 seconds; this signal was unique for each user and was collected from sensors which were placed inside the building. Researchers used infrared technology to achieve the communication between Active Badges and sensors. They also used a component integrated with the Active Badge which automatically turned on and off the device depending on the level of luminosity for battery conservation. Sensors in turn were connected with a master server which was responsible for data processing and database updates. The concept was clear, to forward the call to the user based on her location. A receptionist could monitor the location of users from a screen and when a phone call was received for a specific user she could accurately forward it to him. The application was processing data from sensors and was providing to a table the name, the room that the user might be and the likelihood of finding somebody at that location in the form of a percentage. If a user has not been sighted for 5 minutes, instead of a percentage the field contained the last time and location at which she was sighted. Table was dynamically updated as the user was moving through the building.

2.3.11 PARCTAB (A)

Researchers at Xerox PARC have implemented many research projects to explore ubiquitous computing; PARCTAB [23] was such an experiment. They tried to design and build a computing system that hosted a plethora of applications which employees of the office were using most frequently, inside the perimeter of the building detached from their desktop PC. The system was based to palm-sized computers (handheld devices), infrared technology and LAN network. Researcher used the identity of the user, the location of the user and other objects such as printers etc. as context to provide the appropriate services to the user. A user that was currently in a meeting could see her emails or another co-worker could be aware of a meeting and try not to disturb the bystanders.

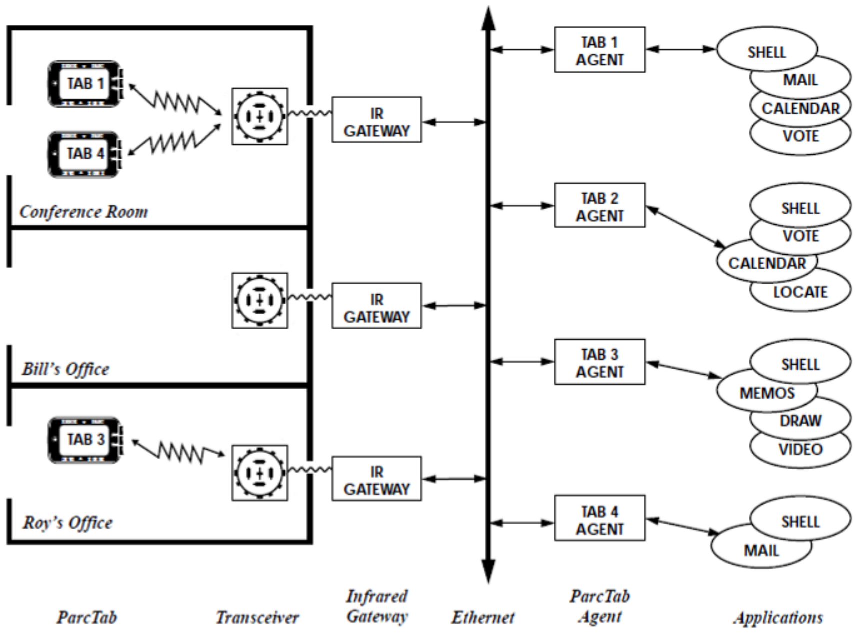


Figure 10 PARCTAB Architecture [23]

“Tabs” were connected via infrared to the transceiver of each room; one or many “tabs” could simultaneously connect to the transceiver. Transceivers were connected via the IP gateway and LAN network to the Tab Agents which were responsible for two main functions: Delivering information from application to the “tab” they served and forwarding messages to the application, again from the bonded device. It is worth mentioned that for each PARCTAB were exactly one tab agent process. From all the applications that users had on their disposal, the most popular were email reader, providing access to e-mail from any place any time, the weather app, file browser app, providing access files stored in the system and the tab loader, which allowed users to download information in the tab’s local memory and use it when needed. In conclusion PARCTAB as elementary context-aware system used specific context in order to provide additional information to the employees to bettering their work environment.

2.3.12 Personal Shopping Assistant (A)

Researchers at AT&T Bell Laboratories developed a wireless system that assisted customers during shopping, the Personal Shopping Assistant [24]. It consists of two components: the PSA device and the PSA server. The handheld device could provide assistance to the shoppers regarding items, prices, point of sales, and so forth through. Information from the PSA server's database was delivered to the user as audio, video or text. If user was registered then a personalized shopping profile was available for her convenience and quality of offered service.

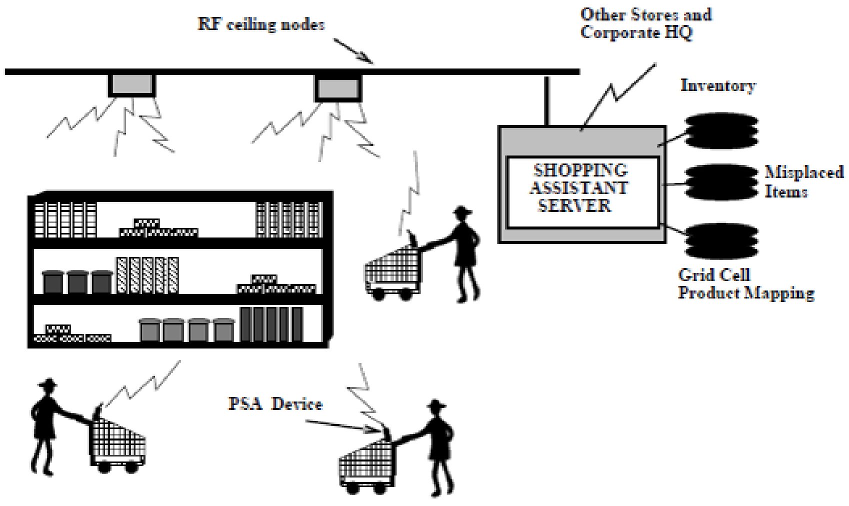

RF ceiling nodes
Other Stores and Corporate HQ
Inventory
SHOPPING ASSISTANT SERVER
Misplaced Items
Grid Cell Product Mapping
PSA Device

Figure 11 Personal Shopping Assistant Architecture [24]

The PSA server is the heart of the PSA system. It consists of five main components: the *position finder* which locates the user, the *Speech Recognition* unit which provides vocabulary recognition, the *Text to Speech* provides the audio streams for the users, the *Customer Database* maintains the profile of every registered customer, the *Connection Manager* is responsible for video and audio channel bandwidth and finally the main process of the PSA server the *Conversation Manager* is responsible for the customer's session and respective services.

2.3.13 Cyberguide (A)

Cyberguide [25] is an application specialized in tourism. The main idea was to provide the opportunity to tourists to have personal devices that can hold throughout the region of a museum, see the exhibits and based on their location to obtain information regarding them. The aspirations of authors were not confined to the museum but also to visit even entire cities with the help of these applications whilst the visitors will have a first

class tour. The researchers of the Georgia Institute of Technology used a series of prototypes applications and handheld devices as tourist guides. They designed two scenarios, one indoor in GVU Center Lab and one outdoor in the Georgia Institute Campus. They used the Apple MessagePad 100 and the Pen based PC which was commercially available. They tried to assign components and services with personalized characters. The main four services were cartographer (notion of physical environment-map), librarian (notion of the information-sightseeing descriptions), navigator (notion of position-navigation of user) and messenger (notion of message exchanging between entities).
In the case of indoor Cyberguide , researchers used a map of the entire GVU Center and also an icon for the location of the user, descriptions for demos or the people involved to these demos, for the communication component they used Appletalk internet connection for receiving or transmitting an email using HTML format. Finally for position component researchers used infrared technology. In case of outdoor Cyberguide they used Campus map, with the corresponding descriptions with the same communication component. The biggest difference was the navigation component which in this case was GPS. Later on they proceed with another outdoor experiment expanding the area, namely a small area of Atlanta.

2.3.14 HEP (A)

HEP [26] is a system that recommends communication services to the caller based on the callee's context. HEP exploit user calendar communication logs to provide sufficient recommendation to the caller. In general the design was based on Microsoft tools .NET, Office etc. More specifically the system consisted of *sensors* (Email, Calendar, Instant Messaging, fixed telephony) which communicated with a *PC client*. Raw data was being retrieved and being subjected to preprocessing on PC client before a *Broker* stored and managed the contextual information. Finally the *Outlook plug-in* provided the UI of the application which helped the user to set her preferences and also monitored the statuses of each of her Outlook contacts. All contacts were publishing their status in order to be publicly available. Depending on the workload that appeared on the Calendar (meetings etc.) or calculating historical data of usage of specific applications (e.g. Word, Excel, PowerPoint) on the computer of each user, an updated user status was shown at the Outlook plug-in and directed the user to the proper mean of communication.

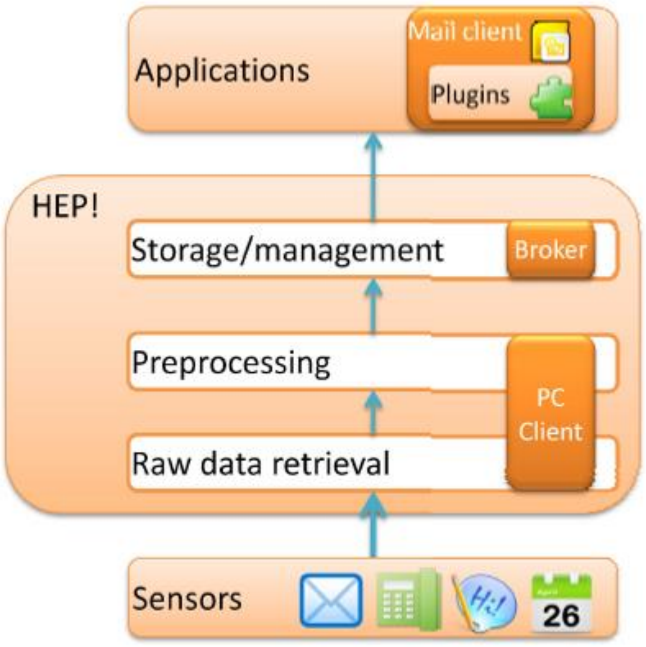


Figure 12 HEP Architecture [26]

2.3.15 Office Assistant (A)

H. Yan and T. Selker designed an agent named Office Assistant [27] that interacts with visitors at the office door and manages the office owner's schedule. As shown in the figure below, the agent is placed to the outer side of the door which the office owner has closed due to noise or a possible meeting. The daily communication of officials is an important factor in a company but this should not create any problems in coworkers' relationships due to inappropriate moments or situations. The communication should be discreet and targeted, a result achieved by the agent. The basic objective of the agents is to offer correct information to the visitor of an office based on the status of office owner; if she has an appointment at the time, if they are busy or if office owner can accept the visitor that moment. If the owner of the office is busy then the agent will notify the visitor in order not to cause discomfort to the meeting in progress. The Office Assistant may create specific models based on the information that has from the office owner, their visitors and the historical visits or the historical agenda of the owner's office. The agent runs on the outdoor computer, which collects context data and interacts with visitors. The internal computer runs a program that communicates with the application installed in the outdoor computer which is responsible for the collection and processing of information that is sent to the office assistant agent. In general the contextual information is consisted of the identity of the visitor, the office owner's schedule status, the office owner's busy status and the office owner's willingness to see the current visitor. Researcher implemented the system using Visual Basic, IBM ViaVoice, Microsoft text-to-speech and Microsoft Outlook (as calendar) to provide contextual information to Office Assistant.

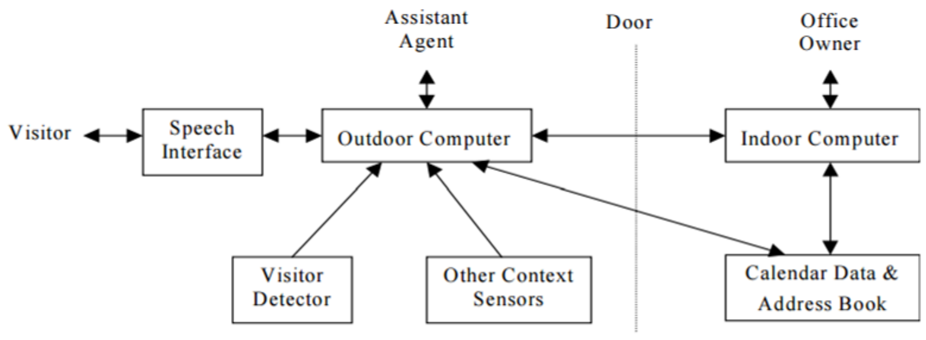


Figure 13 Office Assistant Architecture [27]

2.3.16 A brief comparison

We should mention that various research efforts have been carried out for classification and categorization of frameworks in order to highlight the advantages and disadvantages of each one. In all their surveys [5, 28, 29, 30] researchers tried to compare context-aware systems and middleware solutions by using different taxonomies based on context reasoning, representation, acquiring, storing or architecture.

In brief we can say that Context Toolkit [31] has introduced standard interfaces such as context widget, aggregators etc. CoBrA [12] has introduced the advantages of using semantic-ontologies to manage the policies for context representation and reasoning. CARISMA [15] has shown, using reflection, how conflict resolution can be handled via rules while final decisions are made to maximize the profit among the participating applications. Gaia [16] highlighted the importance of employing multiple reasoning techniques. Aura [18] has shown the importance of having a middleware running seamlessly over many operating systems and devices under different environments and resources. COPAL [19] presented the features of a middleware that relates to IoT, such as loosely coupled plugin architecture, CA4IOT [20] can act as either a standalone middleware or an auxiliary technique to be integrated with other framework solutions and finally SeCoMan [21] SeCoMan proposed a hybrid model which combines user-defined rules via Semantic Web Rule Language (SWRL) as well as description logics. A further analysis of IoT is considered to be out of the scope of this dissertation due to the fact that IoT is a huge research area that uses many different data sources and interfaces that only a part of them contain the usage of smart devices. Our intention is to focus on particular sub-field of IoT, namely the mobile context-awareness with a future vision, the integration of user’s profile in a larger ,more open architecture that contains other components that are disconnected until now such as residence, work environment, shopping centers etc.

From application aspect, most of the aforementioned existing applications used a limited type of information about context, such as location, time, date, identity and static information as some survey highlighted [32]. The reason for using limited information context was mainly the difficulty of having the software systems in the collection and processing of context. Most applications developed for academic purposes in research laboratories. However over the years there have appeared more sophisticated applications that have outgrown the older difficulties and use a combination of multiple sources of information from different mediums, such as sensors or other applications or other web APIs. Google, Yahoo, Apple and other companies have developed many applications and platforms with a wide spectrum that varies from transport (WAZE[1], Google Maps[2], Here[3]) to health care (Apple Health[4], Google Fit[5]) and recommendation (TripAdvisor[6], Booking[7]) to smart homes/offices (Nest[8]).

2.3.17 Summary

In this chapter we examined various frameworks and prototype applications which helped application developer in their further research and developing convenient applications. Based on the aforementioned we will try to examine how to exploit the knowledge of the related work in order to create a logical architecture and a prototype application for user’s profile analysis depending on contextual information.

[1] https://www.waze.com/

[2] https://www.google.gr/maps

[3] https://www.here.com

[4] http://www.apple.com/ios/whats-new/health/

[5] https://fit.google.com/

[6] http://www.tripadvisor.com

[7] http://www.booking.com/

[8] https://nest.com/

3 Problem Definition

In this chapter we will focus on the analysis of the methodology that we will use, with final aim the implementation and completion of this diploma thesis. Afterward we will analyze the architectural approach of the system for development of a Profile Analyzer mobile prototype and finally we will refer to the tools that will help us to accomplish our targets.

3.1 Problem Approach

As we have briefly mentioned in the previous chapters the purpose of this dissertation is to create a prototype mobile application to exploit the contextual information in order to create a context-aware profile analyzer of the user. The application will be developed in the Android Framework. Through specific techniques that we intent to develop, we will try to retrieve as much as possible raw data from sensors namely GPS Sensor, Accelerometer Sensor, Gyroscope Sensor and Light Sensor. With the help of context adapters the raw data will be stored in the database (SQLite Database) which is embedded in the operating system of the mobile phone. Furthermore, we will try to fetch contextual information from the call logger, SMS logger, Google calendar and the launched or installed applications of the mobile device and store them in the database respectively. Afterwards we will endeavor to analyze the data and find conditions that are repeated, recognize patterns that can be used to create rules as for example "When user is in meeting turn the mobile profile to vibration" or "When user finishes work, provide him the all the events that take place in the region of action." Due to the openness of the android framework, several rules can be created which could affect the device functionalities (reducing screen brightness when the battery life is below a specific percentage) based on the context of the user.

Summarizing the steps to follow for the development of the application is initially the creation of all the necessary modules that will capture and store the raw data of the main sensors of mobile devices and then provide a preprocessing data modules in order to produce the required information that can be used to create rules. Secondly, the use of high level context of communication logs (calls, SMS, Skype, Viber depending on the availability and internal security of these applications). Thirdly, the creation of rules and the management of these rules will result in a prototype application that will analyze the context of the profile of the user.

3.2 Methodology

In the context of this dissertation we will attempt to follow the model Design Science Research Methodology which Henver et al [33] noted that creates and evaluates structured IT artifacts intended to solve identified organizational problems. Takeda et al. [34] proposed a design process model which consists of 5 basic process steps: awareness of the problem, suggestion, development, evaluation and conclusion.

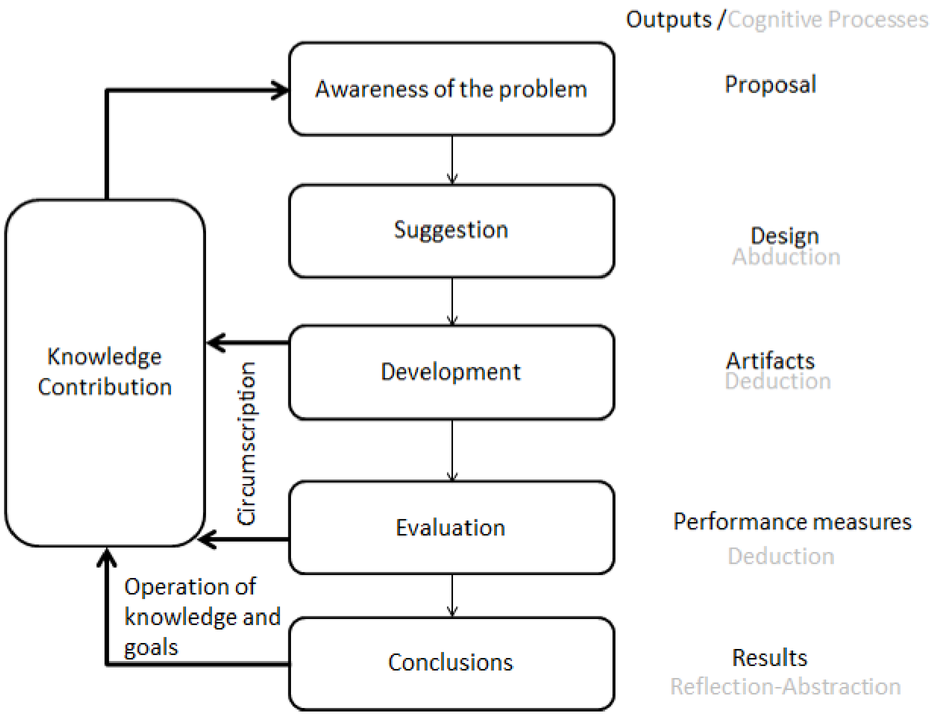


Figure 14 Design Science Research Process Model [34]

Awareness of the problem: The selection of a problem that it is not yet resolved or it is created by new technologies. The outcome of this phase is a proposal for a design model. In this diploma thesis we will attempt to solve a problem which is reflected to the following question: how to create a system in order to analyze the profile of a mobile user through the contextual information of the mobile device?

Suggestion: It refers to the key concepts that are needed in order to solve the problem. In this phase new functionalities or design prototyped are developed aiming at finding a solution for the proposal. Furthermore abduction [35] is used for deriving knowledge from previous related work of the problem area, as we have already presented in the previous chapter. In the remainder of this chapter, we will introduce the architecture and the tools that will assist us in solving the problem.

Development: In this phase the design is developed and implemented with the ultimate goal the creation of an artifact. Several techniques (creation of an algorithm or development of a software module etc.) can be used depending on the kind of the artifact to be produced. In case a new problem appears during the development, this problem will become the new problem that need to be solved in a new iteration of the design cycle. In our attempt the development process will be presented in Chapter 4.

Evaluation: Once an artifact is being developed, the evaluation must take place in order to examine if the created artifact meets the requirements of the proposal. A thoroughly examination is necessary in order to find deviation or failures. In case of contradictions or new findings a new iteration of the design cycle (Circumscription) will take place. In Chapter 5 of the dissertation the evaluation process is presented.

Conclusions: In most of the times this phase is the end of the design process. It is the phase where useful conclusions coming out of the design whether the results are satisfying or with small deviations from the initial hypothetic predictions. However there are several cases where the design cycle needs to be repeated, enhanced with the new acquired knowledge of previous iterations (operation of knowledge and goals). In the final chapter of this dissertation the conclusions of our research are presented.

3.3 Architectural Approach

The architecture will follow is influenced by the Context Toolkit [31] and SOCAM[13] which were presented in Chapter 2 in Related Works. The architecture is based on a four-leveled model with distinct physical blocks in order to create a flexible and modular prototype application. By separating the prototype architecture into tiers, we obtain the option of modifying or adding a new layer or module in the future, instead of reconstructing the entire prototype. The first level consists of sensors that provide the raw data which need a series of preprocesses to become sources of information. The second level includes the entire corresponding context adapter for the retrieval and the storage of raw data from the sensors in the database. Additionally it also includes the high level information providers such as the call logs, Sms and application adapters. Database helper also provides the interconnection between the database and the mobile prototype application. The Application level is the level where the main reasoning and context processes are taking place. This level is the core of the prototype architecture due to the fact that includes and integrates the rules, the processed data, and the context management needed to provide a context-awareness application to the user. The final level is the Presentation layer which interacts with the user via Activities (namely the GUI) or application services that run at the background. It is worth mentioning that all the external API's will connect via web services (SOAP and/or REST) to Context manager in order to minimize the operational cost and decrease the transactions volume from and to database. Below there is an illustration of discussed architecture.

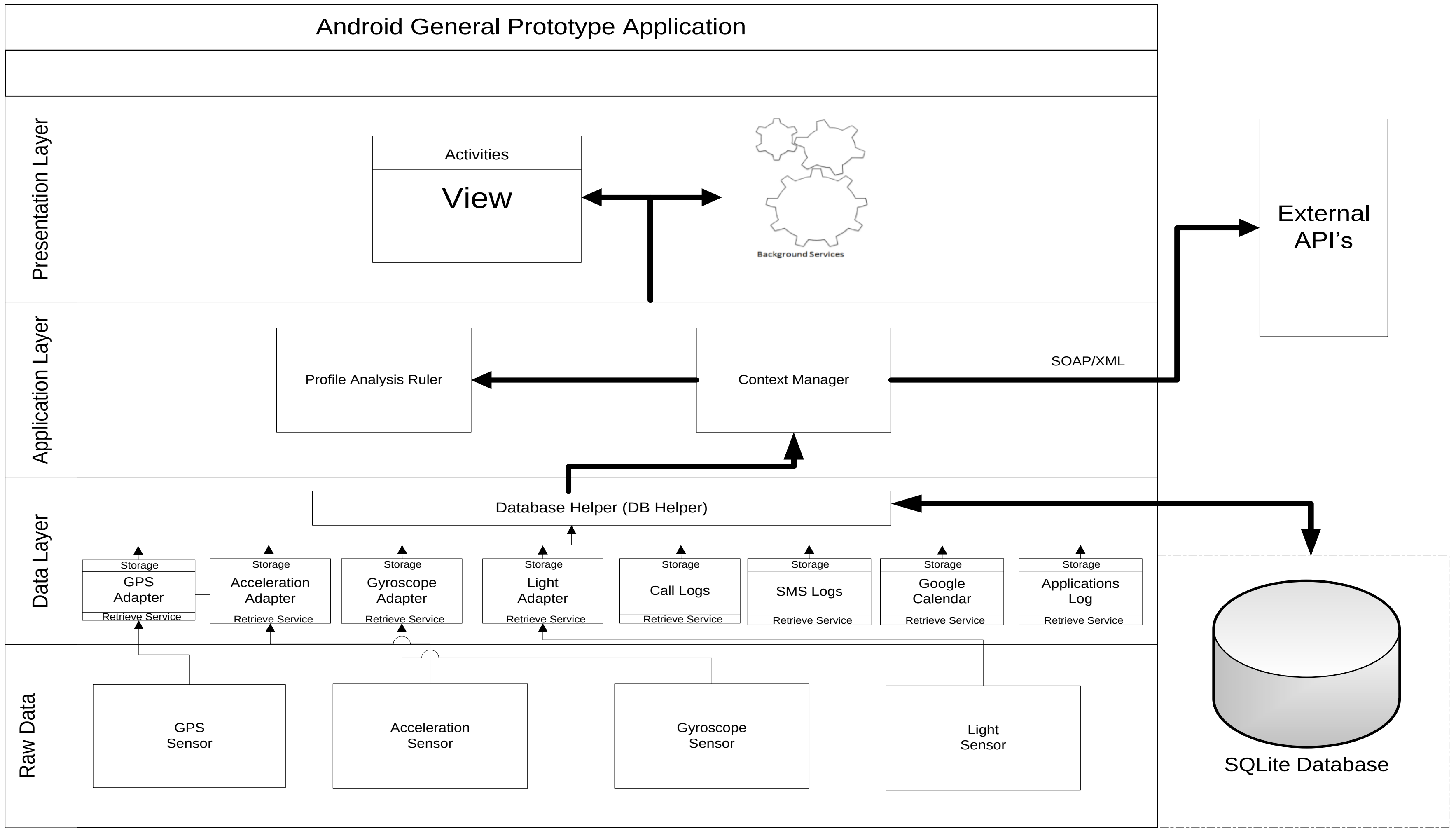


Figure 15 General Architecture of Profile Analyzer

3.4 Tools

We will develop a prototype application for android mobile devices due to the fact that Android operation system is an open source system with a highly customizable application framework. The ecosystem of Android is enormous and consists of many developing tools, a well-documented community and a market place which in 2014 had the 77.8% worldwide market as Gartner reports noted. Another good reason why we will develop an Android prototype their operating system also support sensors such as gyroscopes, accelerometers, proximity sensors, light sensors, weather related sensors that are fully customized and can be used as the developers wish. The application framework is written in Java programming language and it has many plugs in such as Eclipse IDE or even its own development environment Android Studio which we will present in the next paragraph. Below is illustrated the android system architecture[9] which is based on Linux operating system kernel, the android libraries layer such as graphics rendering engine, the android runtime, the application framework that provides classes and design patterns to build applications for android and finally the android applications.

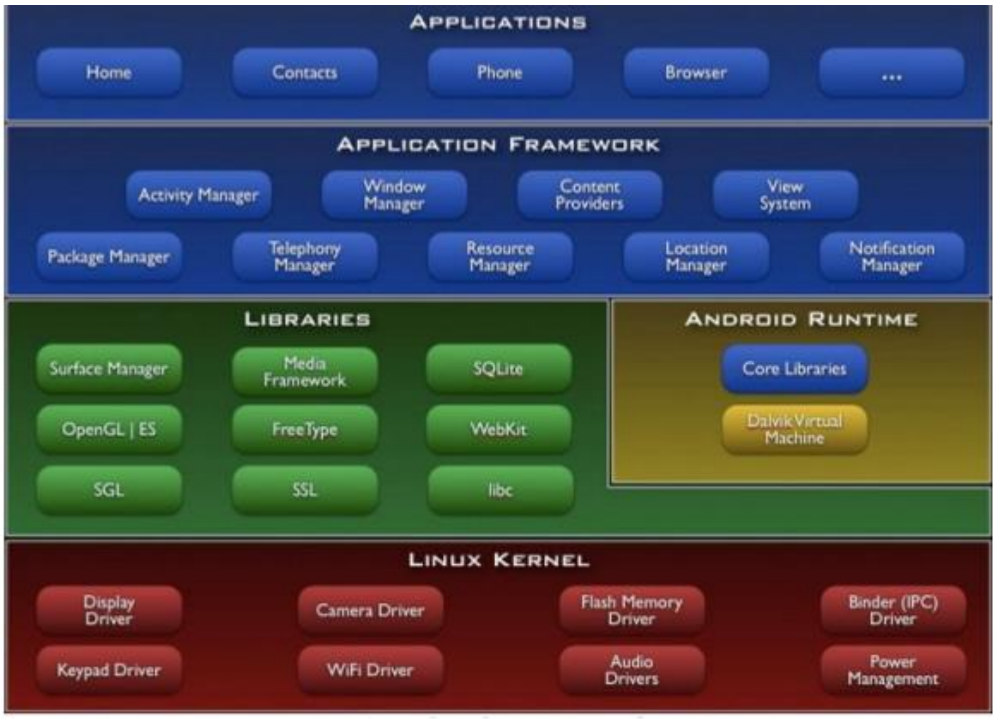


Figure 16 Android System Architecture[10]

[9] https://source.android.com

[10] Http://developer.android.com/images/

3.4.1 Android Studio

Android Studio is an integrated programming environment (IDE) for application development on the Android platform. On May 16, 2013 Google announced a first version of a new environment for android development, a second version was released in June 2014 but the first stable version came out in December 2014 and now the current stable version is 1.3.1 which was released in August 2015. Android Studio is based on software JetBrains' IntelliJ IDEA and was designed exclusively for Android development. It is available for Windows, Mac OS X and Linux. As the majority of IDE, Android Studio allow the creation of a project, writing and editing the code, providing assistant through the GUI, building the project, emulating the project on a variety of devices with different android API’s. [36]

3.4.2 SOAP web services

The prototype application must communicate to the outer word in order to get information regarding the public transports and also events that occurred in a selected area. The implementation of this communication will be achieved by SOAP web services. SOAP, acronym for *Simple Object Access Protocol*, [37] is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks. It relies on Extensible Markup Language (XML) for its message format, and usually relies on other Application Layer protocols, most notably Hypertext Transfer Protocol (HTTP) and Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission. A SOAP message is an ordinary XML document containing the following elements:

- Envelope: Defines the start and the end of the message. It is a mandatory element;
- Header: Contains any optional attributes of the message used in processing the message, either at an intermediary point or at the ultimate end-point;
- Body: Contains the XML data comprising the message being sent. It is a mandatory element;
- Fault: An optional Fault element that provides information about errors that occur while processing the message.

3.4.3 Emulators

During the development period we will create the prototype based on the LG Nexus 5 device. We consider that the choice of this device is the optimal because we avoid memory and performance issues while the application is running as opposed to the virtual emulators and also this specific device has a multitude of sensors. The device's operating system is Android 4.4 (during thesis the OS updated to 5 and 5.1) and also has many sensors such as compass/magnetometer, proximity sensor, accelerometer, light sensor, gyroscope, barometer etc. that are essential for our prototype application.

Figure 17 Nexus 5 Emulator

4 Contribution

In this chapter we will analyze the development of the application that contains separate components that implements a specific function autonomously. The components consist of one class that defines the properties and the methods of an object, a ContextListener who acknowledges the change of the status of a sensor and triggers the procedures that are provided by DBHelper, an Adapter that collects the data from a list and stores them in specific view for later use and finally an activity that call DBHelper procedures and its Adapter in order to present the stored data in a format accustomed to the user. Each adapter and each activity is bound with a layout xml file which implements the corresponding user interface.

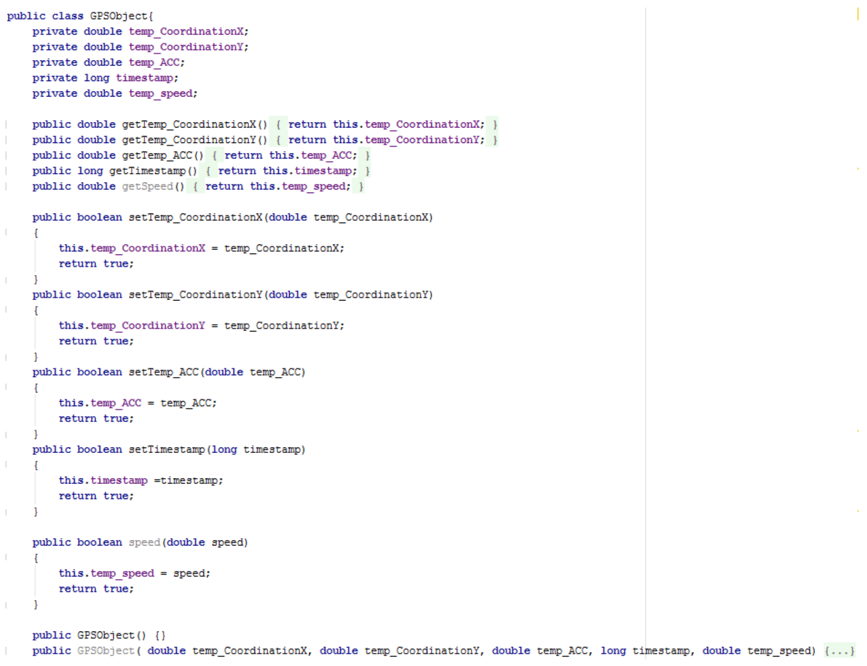
4.1 Implementation

4.1.1 Location Module

This module uses Google's LocationManager [38] and requires specific permissions from the device that a user must provide; namely ACCESS_COARSE_LOCATION or ACCESS_FINE_LOCATION. The data a developer can retrieve (and we are interested in) is latitude and longitude in degrees, the accuracy of the location in meters, the speed in m/s and the timestamp of the coordination.

GPSObject

It consists of four variables of which the first mentioned in longitude, the second mentioned in latitude, the third in the accuracy of positioning and the fourth is the timestamp. Along with the variable definition Set and Get methods are implemented as a pattern of data encapsulation. Instead of accessing class member variables directly, we define get methods to access these variables, and set methods to modify them. Finally two constructors are implemented, the default and a variation with all the proper parameters.

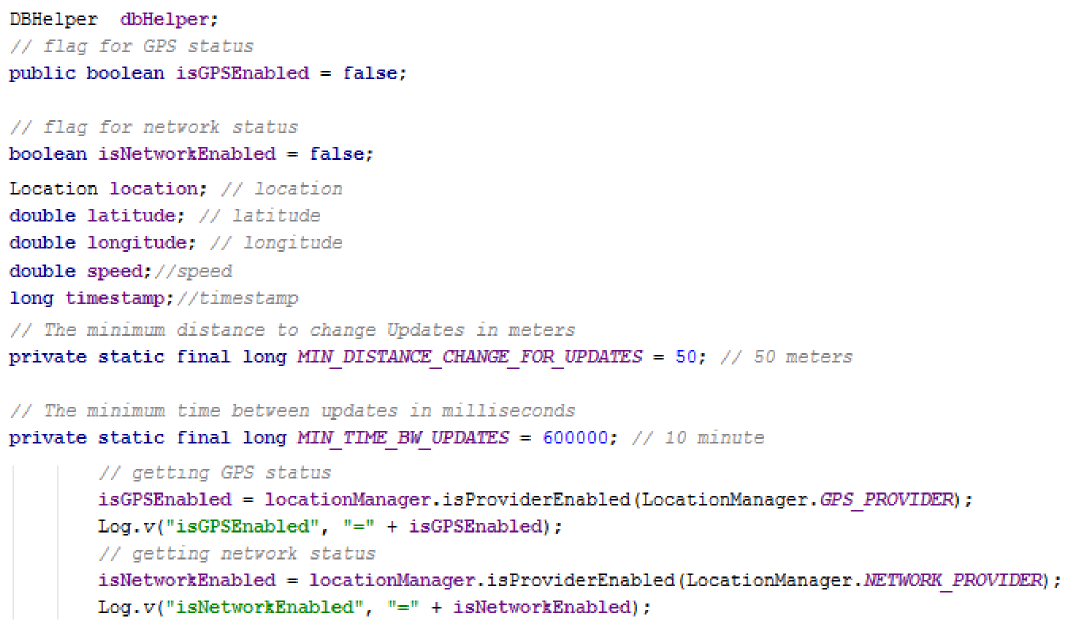


```
public class GPSObject{
    private double temp_CoordinationX;
    private double temp_CoordinationY;
    private double temp_ACC;
    private long timestamp;
    private double temp_speed;

    public double getTemp_CoordinationX() { return this.temp_CoordinationX; }
    public double getTemp_CoordinationY() { return this.temp_CoordinationY; }
    public double getTemp_ACC() { return this.temp_ACC; }
    public long getTimestamp() { return this.timestamp; }
    public double getSpeed() { return this.temp_speed; }

    public boolean setTemp_CoordinationX(double temp_CoordinationX)
    {
        this.temp_CoordinationX = temp_CoordinationX;
        return true;
    }
    public boolean setTemp_CoordinationY(double temp_CoordinationY)
    {
        this.temp_CoordinationY = temp_CoordinationY;
        return true;
    }
    public boolean setTemp_ACC(double temp_ACC)
    {
        this.temp_ACC = temp_ACC;
        return true;
    }
    public boolean setTimestamp(long timestamp)
    {
        this.timestamp =timestamp;
        return true;
    }

    public boolean speed(double speed)
    {
        this.temp_speed = speed;
        return true;
    }

    public GPSObject() {}
    public GPSObject( double temp_CoordinationX, double temp_CoordinationY, double temp_ACC, long timestamp, double temp_speed) {...}
```

Figure 18 GPSObject Class

GPSListener

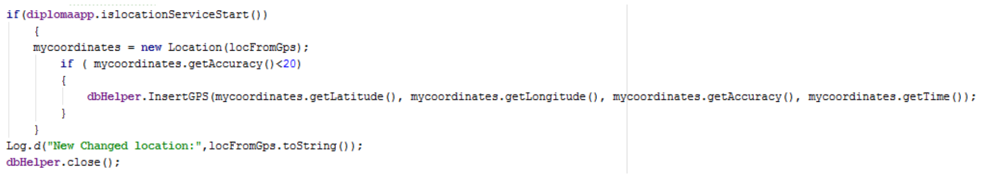
GpsListener implements the class LocationListener [39] which is used for receiving notifications from the LocationManager when the location has changed. It checks the status of the network and the availability of the GPS sensor on the mobile phone and updates the status of the sensor every 10 minutes or if the distance value has changed by 50 meters. Additionally it defines a DBHelper object which is responsible for the data management and several other necessary parameters.



```
DBHelper  dbHelper;
// flag for GPS status
public boolean isGPSEnabled = false;

// flag for network status
boolean isNetworkEnabled = false;

Location location; // location
double latitude; // latitude
double longitude; // longitude
double speed;//speed
long timestamp;//timestamp
// The minimum distance to change Updates in meters
private static final long MIN_DISTANCE_CHANGE_FOR_UPDATES = 50; // 50 meters

// The minimum time between updates in milliseconds
private static final long MIN_TIME_BW_UPDATES = 600000; // 10 minute
        // getting GPS status
        isGPSEnabled = locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER);
        Log.v("isGPSEnabled", "=" + isGPSEnabled);
        // getting network status
        isNetworkEnabled = locationManager.isProviderEnabled(LocationManager.NETWORK_PROVIDER);
        Log.v("isNetworkEnabled", "=" + isNetworkEnabled);
```

Figure 19 GPSListener - Parameters

In case network is available then LocationManager provides the coordinates from the network to the GPSListener whereas in case GPS is available LocationManager provides the coordinates from GPS. We defined an accuracy of 20 meters as threshold in order to discard invalid data. If the accuracy of GPS is below 20 meters then GPSListener calls the *InsertGPS* procedure from DBHelper object and store the coordinates in the database. The above values can be changed during tests.

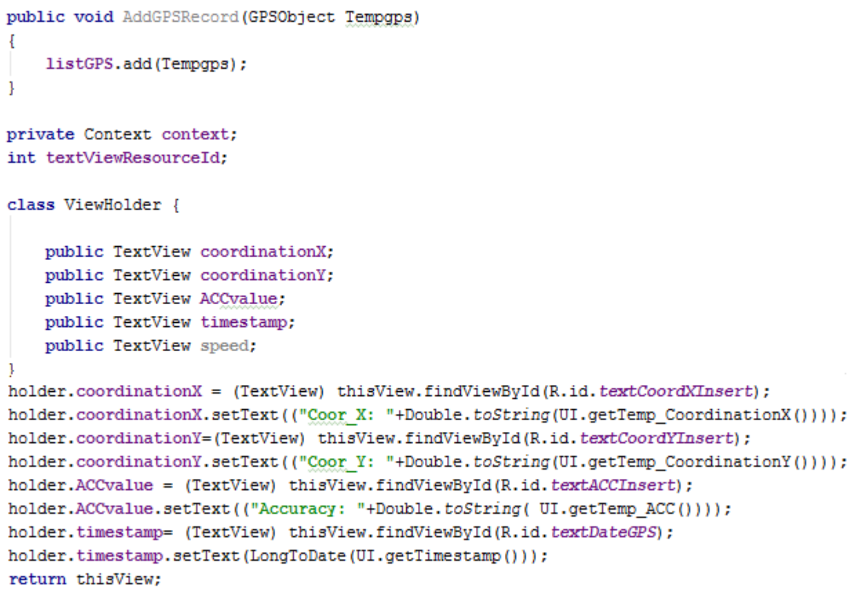


```
if(diplomaapp.islocationServiceStart())
    {
    mycoordinates = new Location(locFromGps);
        if ( mycoordinates.getAccuracy()<20)
        {
            dbHelper.InsertGPS(mycoordinates.getLatitude(), mycoordinates.getLongitude(), mycoordinates.getAccuracy(), mycoordinates.getTime());
        }
    }
Log.d("New Changed location:",locFromGps.toString());
dbHelper.close();
```

Figure 20 GPSListener – Insertion of new coordinates

GPSAdapter

GPSAdapter is a class that extends an array adapter that is filled with a list of *GPSObjects*. It connects the potential GPS data with a list view, in other words we define the look of the list and prepare the specific list in case is required by the application. This is achieved via a Viewholder class in which we define the corresponding values of the *GPSObject* object and bind them with the appropriate list view that we have already created based on the *R.id* identity.

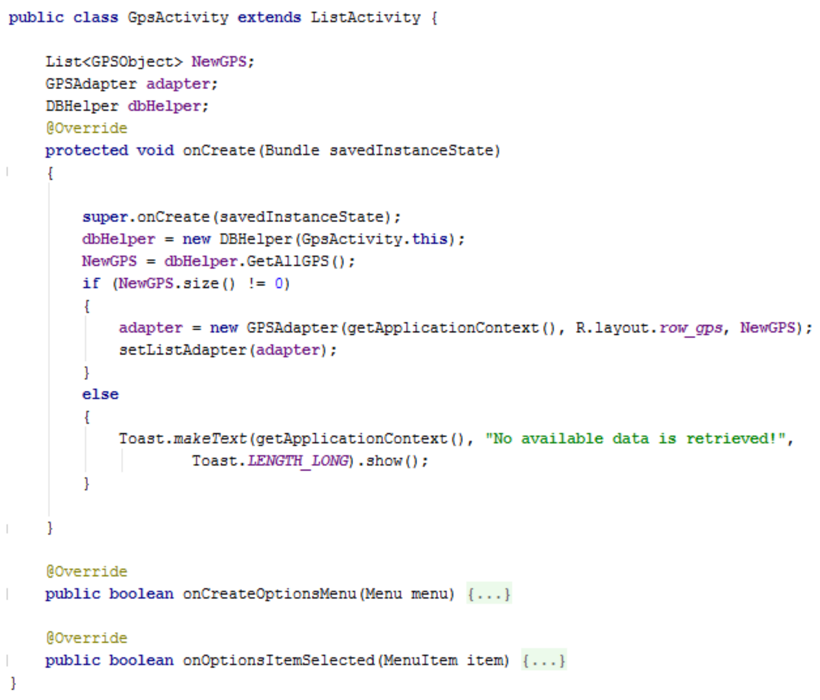


```
public void AddGPSRecord(GPSObject Tempgps)
{
    listGPS.add(Tempgps);
}

private Context context;
int textViewResourceId;

class ViewHolder {

    public TextView coordinationX;
    public TextView coordinationY;
    public TextView ACCvalue;
    public TextView timestamp;
    public TextView speed;
}
holder.coordinationX = (TextView) thisView.findViewById(R.id.textCoordXInsert);
holder.coordinationX.setText(("Coor_X: "+Double.toString(UI.getTemp_CoordinationX())));
holder.coordinationY=(TextView) thisView.findViewById(R.id.textCoordYInsert);
holder.coordinationY.setText(("Coor_Y: "+Double.toString(UI.getTemp_CoordinationY())));
holder.ACCvalue = (TextView) thisView.findViewById(R.id.textACCInsert);
holder.ACCvalue.setText(("Accuracy: "+Double.toString( UI.getTemp_ACC())));
holder.timestamp= (TextView) thisView.findViewById(R.id.textDateGPS);
holder.timestamp.setText(LongToDate(UI.getTimestamp()));
return thisView;
```

Figure 21 GPSAdapter - Binding view and object

GPSActivity

This activity is a complementary module to provide user with a visualization of data that the application is stored for later use. We consider that this display is important for the sense of security that user must have in order to be aware each moment what information is accessed by the application. The class extends the ListActivity class and inside the class we define a DBHelper object and a list of GPSObjects which are filled in with the GPS records from DBHelper procedure *GetAllGPS*.



```java
public class GpsActivity extends ListActivity {

    List<GPSObject> NewGPS;
    GPSAdapter adapter;
    DBHelper dbHelper;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {

        super.onCreate(savedInstanceState);
        dbHelper = new DBHelper(GpsActivity.this);
        NewGPS = dbHelper.GetAllGPS();
        if (NewGPS.size() != 0)
        {
            adapter = new GPSAdapter(getApplicationContext(), R.layout.row_gps, NewGPS);
            setListAdapter(adapter);
        }
        else
        {
            Toast.makeText(getApplicationContext(), "No available data is retrieved!",
                    Toast.LENGTH_LONG).show();
        }

    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {...}

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {...}
}
```

Figure 22 GPSActivity - Display the available GPS records

4.1.2 Acceleration Module

This module uses Google’s Sensor [40] with SensorManager, SensorEventListener and SensorEvent. The data a developer can retrieve (and we are interested in) is the accelerator from all axis (x, y, z) and the timestamp of the event. As the SensorEvent API [41] notes the X axis is horizontal and points to the right, the Y axis is vertical and points up and the Z axis points towards the outside of the front face of the screen. In this system, coordinates behind the screen have negative Z values. All values are in SI units (m/s^2) having deducted the gravitational force G_y.

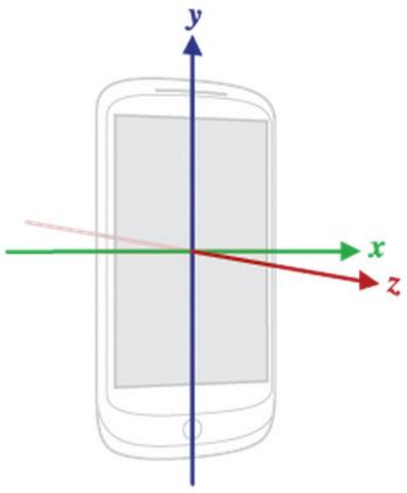
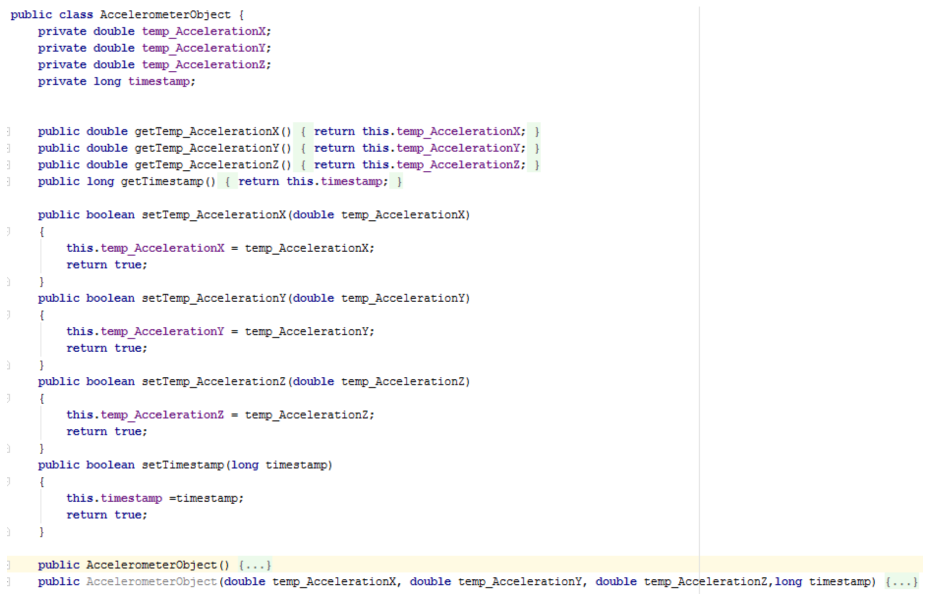


Figure 23 Accelerometer coordinate system [41]

AccelerometerObject

It consists of four variables of which the first mentioned in acceleration in x-axis, the second mentioned in acceleration in y-axis, the third in acceleration in z-axis and the fourth is the timestamp. Along with the variable definition Set and Get methods are implemented as a pattern of data encapsulation. Instead of accessing class member variables directly, we define get methods to read these variables, and set methods to modify them. Finally two constructors are implemented, the default and a variation with all the proper parameters.

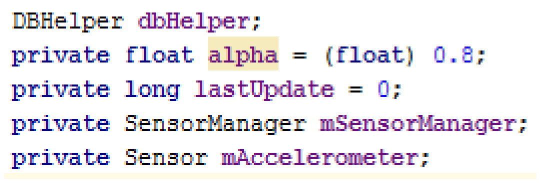


```
public class AccelerometerObject {
    private double temp_AccelerationX;
    private double temp_AccelerationY;
    private double temp_AccelerationZ;
    private long timestamp;


    public double getTemp_AccelerationX() { return this.temp_AccelerationX; }
    public double getTemp_AccelerationY() { return this.temp_AccelerationY; }
    public double getTemp_AccelerationZ() { return this.temp_AccelerationZ; }
    public long getTimestamp() { return this.timestamp; }

    public boolean setTemp_AccelerationX(double temp_AccelerationX)
    {
        this.temp_AccelerationX = temp_AccelerationX;
        return true;
    }
    public boolean setTemp_AccelerationY(double temp_AccelerationY)
    {
        this.temp_AccelerationY = temp_AccelerationY;
        return true;
    }
    public boolean setTemp_AccelerationZ(double temp_AccelerationZ)
    {
        this.temp_AccelerationZ = temp_AccelerationZ;
        return true;
    }
    public boolean setTimestamp(long timestamp)
    {
        this.timestamp =timestamp;
        return true;
    }

    public AccelerometerObject() {...}
    public AccelerometerObject(double temp_AccelerationX, double temp_AccelerationY, double temp_AccelerationZ,long timestamp) {...}
```

Figure 24 AccelerometerObject Class

AccelerometerListener

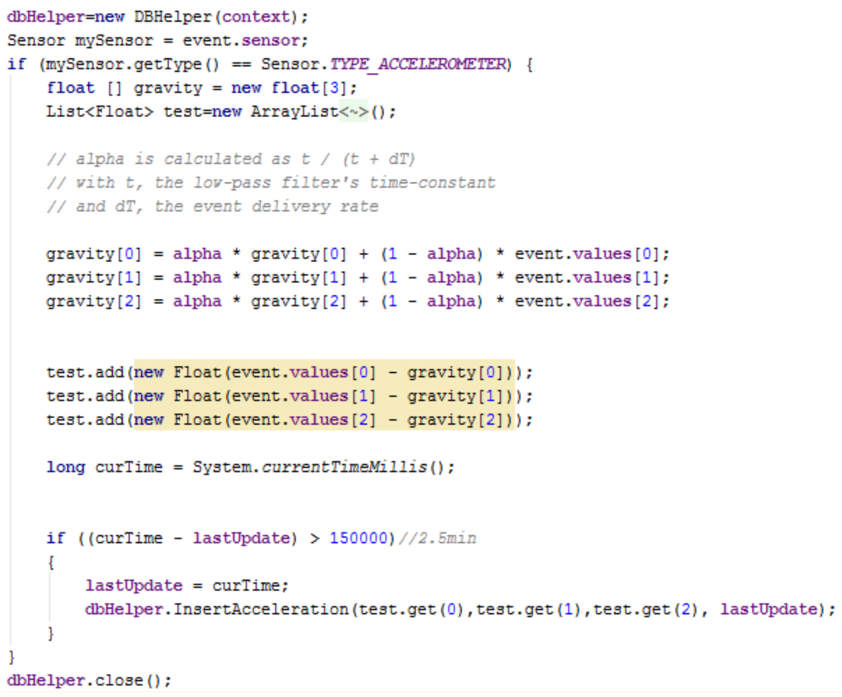
AccelerometerListener implements the class SensorEventListener [42] which is used for receiving notifications from the SensorManager when the acceleration (sensor. TYPE_ACCELEROMETER) has changed. We define a DBHelper object for the database connection and SensorManager for the access of device's accelerometer sensor.



```
DBHelper dbHelper;
private float alpha = (float) 0.8;
private long lastUpdate = 0;
private SensorManager mSensorManager;
private Sensor mAccelerometer;
```

Figure 25 AccelerometerListener – Parameters

When this class is called by the system, it registers the sensor manager event in order to get the values of the sensor. If the last update of the sensor is greater than 2.5 minutes (this value can change during tests) then AccelerometerListener calls the *InsertAcceleration* procedure from DBHelper object and store the values in the database. A low-pass filter is used to isolate the force of gravity as the figure below illustrates.

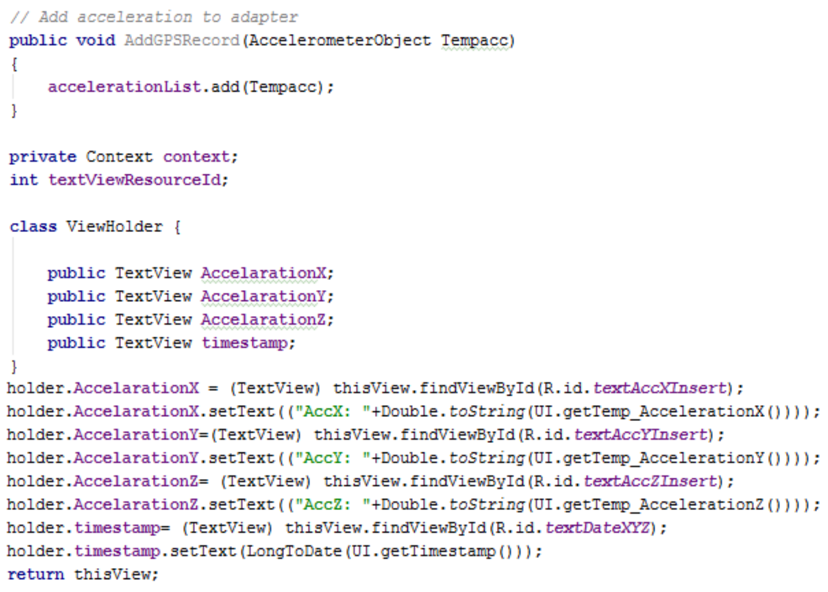


```
dbHelper=new DBHelper(context);
Sensor mySensor = event.sensor;
if (mySensor.getType() == Sensor.TYPE_ACCELEROMETER) {
    float [] gravity = new float[3];
    List<Float> test=new ArrayList<~>();

    // alpha is calculated as t / (t + dT)
    // with t, the low-pass filter's time-constant
    // and dT, the event delivery rate

    gravity[0] = alpha * gravity[0] + (1 - alpha) * event.values[0];
    gravity[1] = alpha * gravity[1] + (1 - alpha) * event.values[1];
    gravity[2] = alpha * gravity[2] + (1 - alpha) * event.values[2];


    test.add(new Float(event.values[0] - gravity[0]));
    test.add(new Float(event.values[1] - gravity[1]));
    test.add(new Float(event.values[2] - gravity[2]));

    long curTime = System.currentTimeMillis();


    if ((curTime - lastUpdate) > 150000)//2.5min
    {
        lastUpdate = curTime;
        dbHelper.InsertAcceleration(test.get(0),test.get(1),test.get(2), lastUpdate);
    }
}
dbHelper.close();
```

Figure 26 AccelerometerListener - Insertion of new values

AccelerometerAdapter

AccelerometerAdapter is a class that extends an array adapter that is filled with a list of *AccelerometerObjects*. It connects the potential acceleration data with a list view, in other words we define the look of the list and prepare the specific list in case is required by the application. This is achieved via a Viewholder class in which we define the corresponding values of the *AccelerometerObject* object and bind them with the appropriate list view that we have already created based on the *R.id* identity.

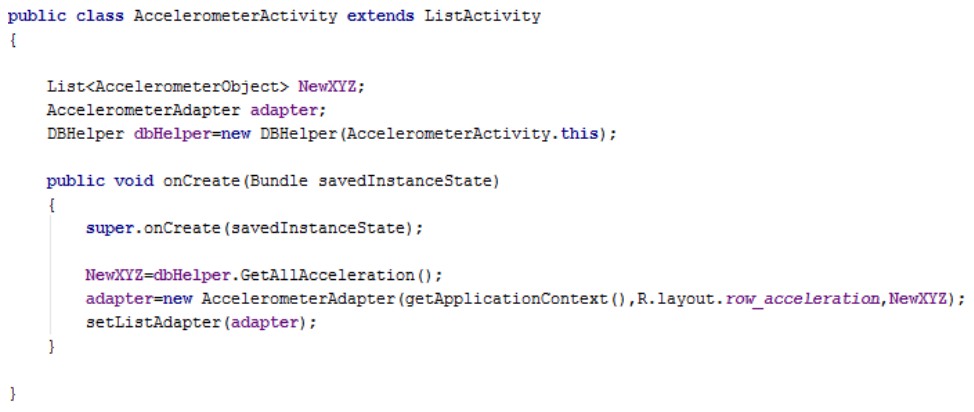


```
// Add acceleration to adapter
public void AddGPSRecord(AccelerometerObject Tempacc)
{
    accelerationList.add(Tempacc);
}

private Context context;
int textViewResourceId;

class ViewHolder {

    public TextView AccelarationX;
    public TextView AccelarationY;
    public TextView AccelarationZ;
    public TextView timestamp;
}
holder.AccelarationX = (TextView) thisView.findViewById(R.id.textAccXInsert);
holder.AccelarationX.setText(("AccX: "+Double.toString(UI.getTemp_AccelerationX())));
holder.AccelarationY=(TextView) thisView.findViewById(R.id.textAccYInsert);
holder.AccelarationY.setText(("AccY: "+Double.toString(UI.getTemp_AccelerationY())));
holder.AccelarationZ= (TextView) thisView.findViewById(R.id.textAccZInsert);
holder.AccelarationZ.setText(("AccZ: "+Double.toString(UI.getTemp_AccelerationZ())));
holder.timestamp= (TextView) thisView.findViewById(R.id.textDateXYZ);
holder.timestamp.setText(LongToDate(UI.getTimestamp()));
return thisView;
```

Figure 27 AccelerometerAdapter - Binding view and object

AccelerometerActivity

This activity is a complementary module to provide user with a visualization of data that the application is stored for later use as the previous activity. The class extends the ListActivity class and inside the class we define a DBHelper object and a list of AccelerometerObjects which are filled in with the acceleration records from DBHelper procedure *GetAllAcceleration*.



```
public class AccelerometerActivity extends ListActivity
{

    List<AccelerometerObject> NewXYZ;
    AccelerometerAdapter adapter;
    DBHelper dbHelper=new DBHelper(AccelerometerActivity.this);

    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        NewXYZ=dbHelper.GetAllAcceleration();
        adapter=new AccelerometerAdapter(getApplicationContext(),R.layout.row_acceleration,NewXYZ);
        setListAdapter(adapter);
    }

}
```

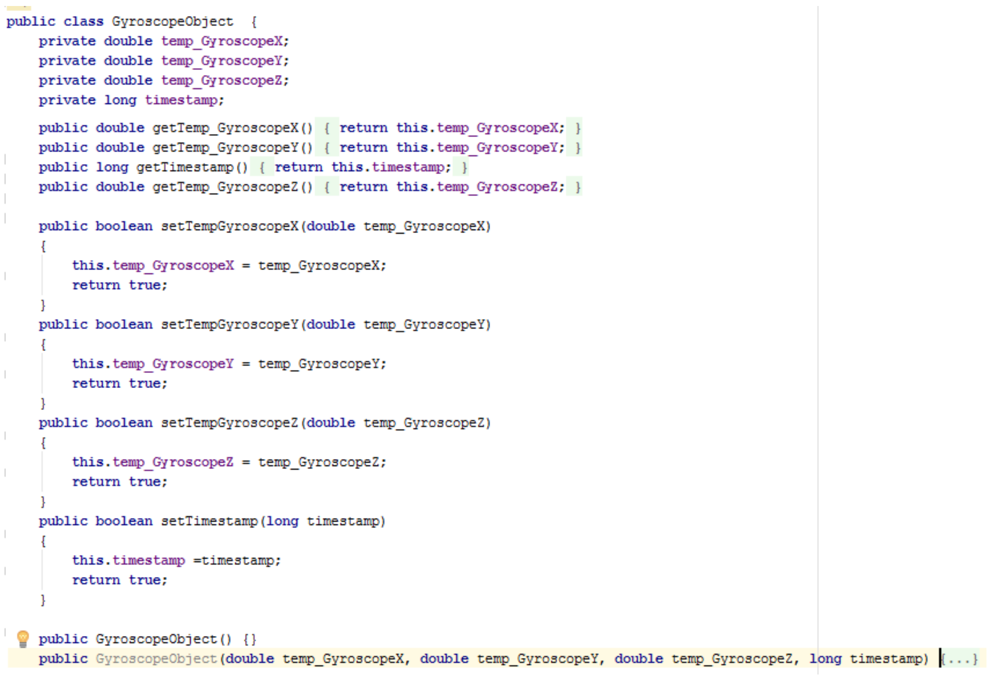
Figure 28 AccelerometerActivity - Display the available accelerometer values

4.1.3 Gyroscope Module

This module uses the same Google Sensor [40] as the Accelerometer Module. An important difference is that SensorManager call the *Sensor.TYPE_GYROSCOPE* instead of the accelerometer. Consequently the returned values are in radians/second and measure the rate of rotation around the device's local X, Y and Z axis as *Figure 23 Accelerometer coordinate system [41]* displays. The values are the angular speed around the three axis.

GyroscopeObject

It consists of four variables of which the first mentioned in gyroscope in x-axis, the second mentioned in gyroscope in y-axis, the third in gyroscope in z-axis and the fourth is the timestamp. Along with the variable definition Set and Get methods are implemented as a pattern of data encapsulation. Instead of accessing class member variables directly, we define get methods to access these variables, and set methods to modify them. Finally two constructors are implemented, the default and a variation with all the proper parameters.

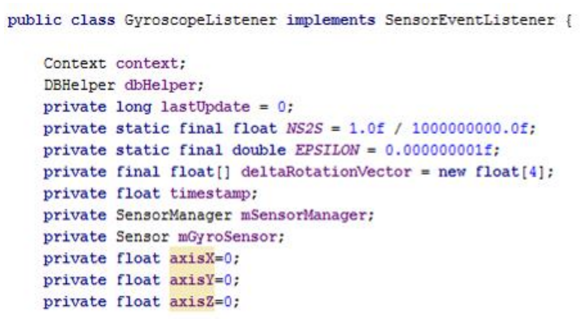


```
public class GyroscopeObject  {
    private double temp_GyroscopeX;
    private double temp_GyroscopeY;
    private double temp_GyroscopeZ;
    private long timestamp;

    public double getTemp_GyroscopeX() { return this.temp_GyroscopeX; }
    public double getTemp_GyroscopeY() { return this.temp_GyroscopeY; }
    public long getTimestamp() { return this.timestamp; }
    public double getTemp_GyroscopeZ() { return this.temp_GyroscopeZ; }

    public boolean setTempGyroscopeX(double temp_GyroscopeX)
    {
        this.temp_GyroscopeX = temp_GyroscopeX;
        return true;
    }
    public boolean setTempGyroscopeY(double temp_GyroscopeY)
    {
        this.temp_GyroscopeY = temp_GyroscopeY;
        return true;
    }
    public boolean setTempGyroscopeZ(double temp_GyroscopeZ)
    {
        this.temp_GyroscopeZ = temp_GyroscopeZ;
        return true;
    }
    public boolean setTimestamp(long timestamp)
    {
        this.timestamp =timestamp;
        return true;
    }

    public GyroscopeObject() {}
    public GyroscopeObject(double temp_GyroscopeX, double temp_GyroscopeY, double temp_GyroscopeZ, long timestamp) {...}
```

Figure 29 GyroscopeObject - Class

GyroscopeListener

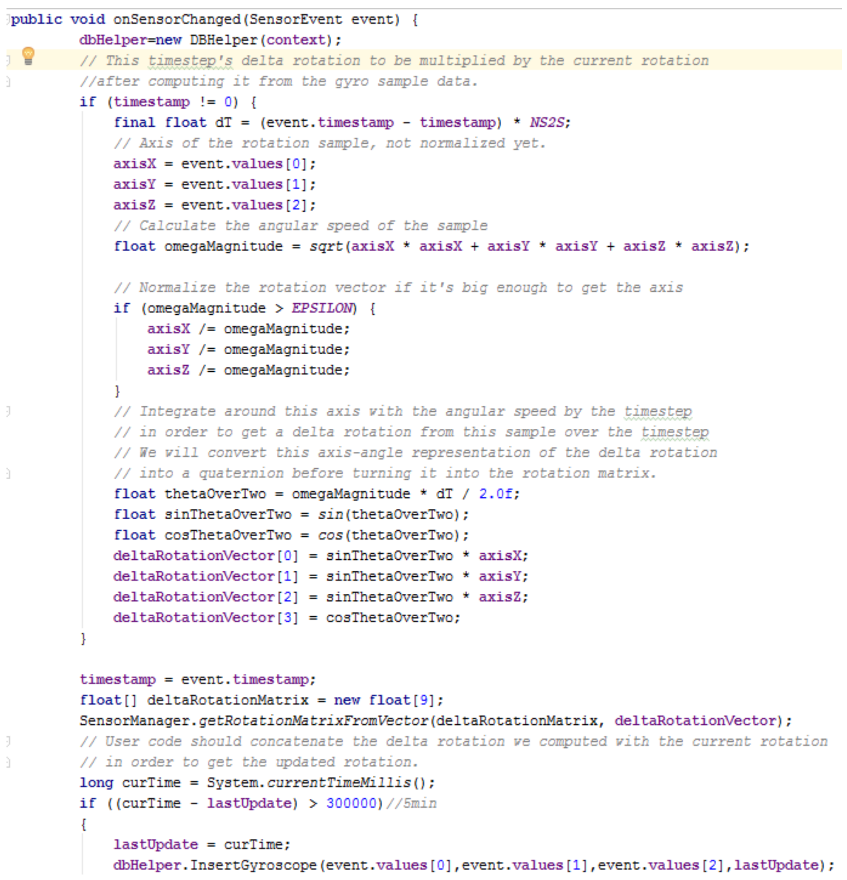
GyroscopesListener implements the class SensorEventListener [41] which is used for receiving notifications from the SensorManager when gyroscope has changed. We define a DBHelper object for the database connection and SensorManager for the access of device's gyroscope sensor.



```java
public class GyroscopeListener implements SensorEventListener {

    Context context;
    DBHelper dbHelper;
    private long lastUpdate = 0;
    private static final float NS2S = 1.0f / 1000000000.0f;
    private static final double EPSILON = 0.000000001f;
    private final float[] deltaRotationVector = new float[4];
    private float timestamp;
    private SensorManager mSensorManager;
    private Sensor mGyroSensor;
    private float axisX=0;
    private float axisY=0;
    private float axisZ=0;
```

Figure 30 GyroscopesListener – Parameters

When this class is called by the system, it registers the sensor manager event in order to get the values of the sensor. If the last update of the sensor is greater than five minutes (this value can change during tests) then GyroscopeListener calls the *InsertGyroscope* procedure from DBHelper object and store the values in the database. Before the storage of the values the gyroscope is integrated over time to calculate a rotation describing the change of angles over a timestep.

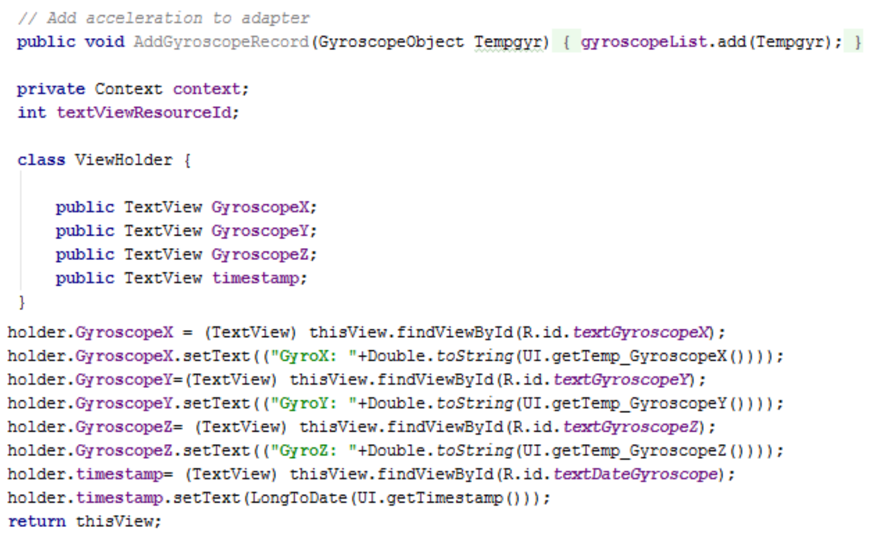


```java
public void onSensorChanged(SensorEvent event) {
        dbHelper=new DBHelper(context);
        // This timestep's delta rotation to be multiplied by the current rotation
        //after computing it from the gyro sample data.
        if (timestamp != 0) {
            final float dT = (event.timestamp - timestamp) * NS2S;
            // Axis of the rotation sample, not normalized yet.
            axisX = event.values[0];
            axisY = event.values[1];
            axisZ = event.values[2];
            // Calculate the angular speed of the sample
            float omegaMagnitude = sqrt(axisX * axisX + axisY * axisY + axisZ * axisZ);

            // Normalize the rotation vector if it's big enough to get the axis
            if (omegaMagnitude > EPSILON) {
                axisX /= omegaMagnitude;
                axisY /= omegaMagnitude;
                axisZ /= omegaMagnitude;
            }
            // Integrate around this axis with the angular speed by the timestep
            // in order to get a delta rotation from this sample over the timestep
            // We will convert this axis-angle representation of the delta rotation
            // into a quaternion before turning it into the rotation matrix.
            float thetaOverTwo = omegaMagnitude * dT / 2.0f;
            float sinThetaOverTwo = sin(thetaOverTwo);
            float cosThetaOverTwo = cos(thetaOverTwo);
            deltaRotationVector[0] = sinThetaOverTwo * axisX;
            deltaRotationVector[1] = sinThetaOverTwo * axisY;
            deltaRotationVector[2] = sinThetaOverTwo * axisZ;
            deltaRotationVector[3] = cosThetaOverTwo;
        }

        timestamp = event.timestamp;
        float[] deltaRotationMatrix = new float[9];
        SensorManager.getRotationMatrixFromVector(deltaRotationMatrix, deltaRotationVector);
        // User code should concatenate the delta rotation we computed with the current rotation
        // in order to get the updated rotation.
        long curTime = System.currentTimeMillis();
        if ((curTime - lastUpdate) > 300000)//5min
        {
            lastUpdate = curTime;
            dbHelper.InsertGyroscope(event.values[0],event.values[1],event.values[2],lastUpdate);
```

Figure 31 GyroscopesListener - Insertion of new values

GyroscopeAdapter

GyroscopeAdapter is a class that extends an array adapter that is filled with a list of *GyroscopeObjects*. It connects the potential gyroscope data with a list view, in other words we define the look of the list and prepare the specific list in case is required by the application. This is achieved via a Viewholder class in which we define the corresponding values of the *GyroscopeObject* object and bind them with the appropriate list view that we have already created based on the *R.id* identity.

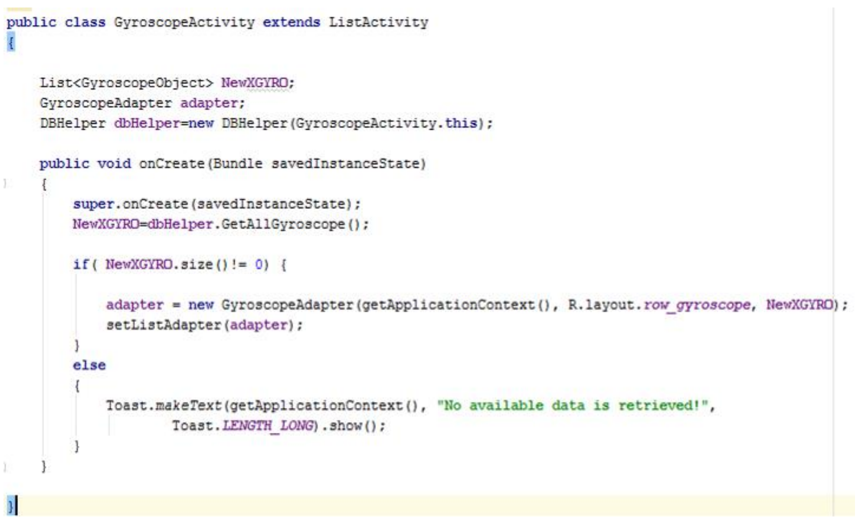


```
// Add acceleration to adapter
public void AddGyroscopeRecord(GyroscopeObject Tempgyr) { gyroscopeList.add(Tempgyr); }

private Context context;
int textViewResourceId;

class ViewHolder {

    public TextView GyroscopeX;
    public TextView GyroscopeY;
    public TextView GyroscopeZ;
    public TextView timestamp;
}
holder.GyroscopeX = (TextView) thisView.findViewById(R.id.textGyroscopeX);
holder.GyroscopeX.setText(("GyroX: "+Double.toString(UI.getTemp_GyroscopeX())));
holder.GyroscopeY=(TextView) thisView.findViewById(R.id.textGyroscopeY);
holder.GyroscopeY.setText(("GyroY: "+Double.toString(UI.getTemp_GyroscopeY())));
holder.GyroscopeZ= (TextView) thisView.findViewById(R.id.textGyroscopeZ);
holder.GyroscopeZ.setText(("GyroZ: "+Double.toString(UI.getTemp_GyroscopeZ())));
holder.timestamp= (TextView) thisView.findViewById(R.id.textDateGyroscope);
holder.timestamp.setText(LongToDate(UI.getTimestamp()));
return thisView;
```

Figure 32 GyroscopeAdapter - Binding view and object

GyroscopeActivity

This activity is a complementary module to provide user with a visualization of data that the application is stored for later use as the previous activity. The class extends the ListActivity class and inside the class we define a DBHelper object and a list of GyroscopeObjects which are filled in with the gyroscope records from DBHelper procedure *GetAllGyroscope*.



```
public class GyroscopeActivity extends ListActivity
{

    List<GyroscopeObject> NewXGYRO;
    GyroscopeAdapter adapter;
    DBHelper dbHelper=new DBHelper(GyroscopeActivity.this);

    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        NewXGYRO=dbHelper.GetAllGyroscope();

        if( NewXGYRO.size()!= 0) {

            adapter = new GyroscopeAdapter(getApplicationContext(), R.layout.row_gyroscope, NewXGYRO);
            setListAdapter(adapter);
        }
        else
        {
            Toast.makeText(getApplicationContext(), "No available data is retrieved!",
                    Toast.LENGTH_LONG).show();
        }
    }

}
```

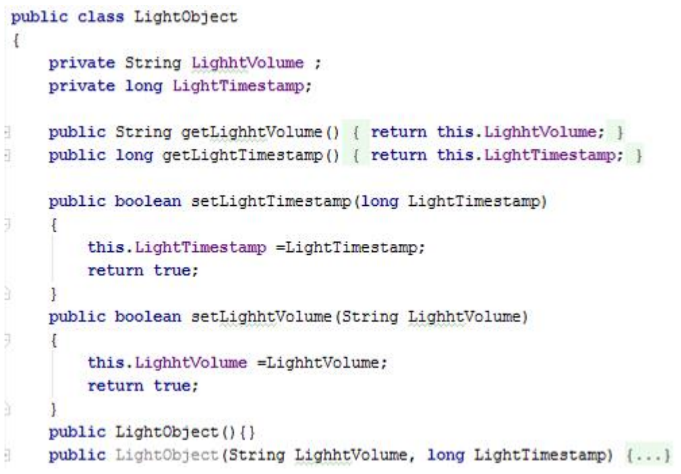
Figure 33 GyroscopeActivity - Display the available gyroscope values

4.1.4 Light Module

This module uses the same Google Sensor as the previous ones. An important difference is that SensorManager call the *Sensor.TYPE_LIGHT*. Consequently the returned value is the ambient light level in lux units.

LightObject

It consists of two variables of which the first mentioned in light volume and the second is the timestamp. Along with the variable definition Set and Get methods are implemented as a pattern of data encapsulation. Instead of accessing class member variables directly, we define get methods to access these variables, and set methods to modify them. Finally two constructors are implemented, the default and a variation with all the proper parameters.

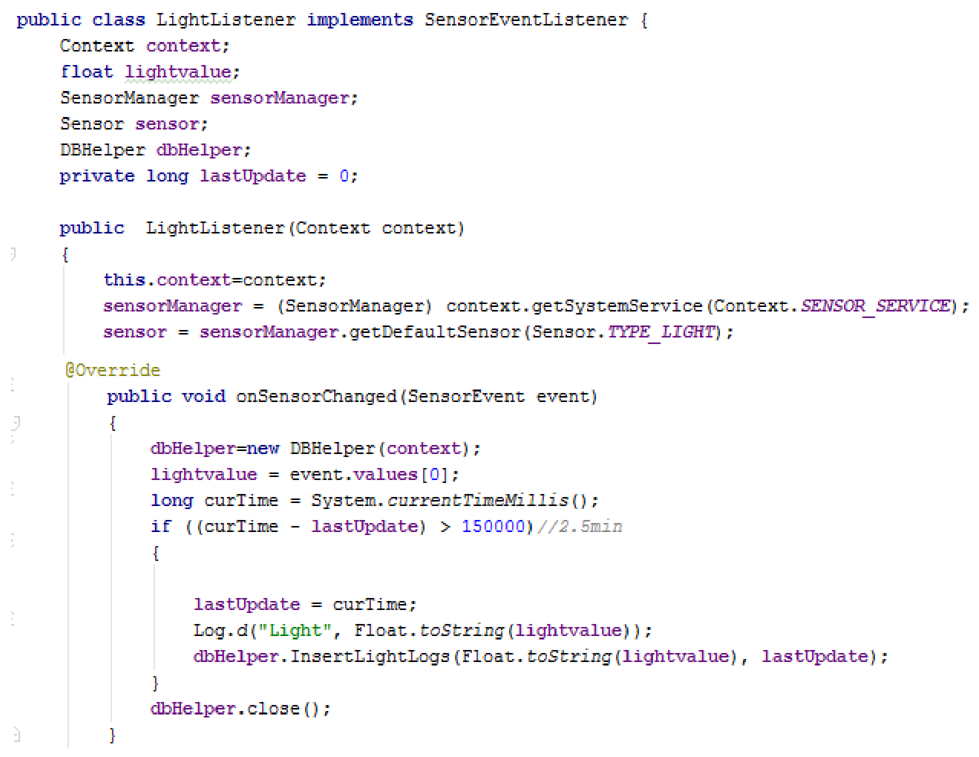


```
public class LightObject
{
    private String LighhtVolume ;
    private long LightTimestamp;

    public String getLighhtVolume() { return this.LighhtVolume; }
    public long getLightTimestamp() { return this.LightTimestamp; }

    public boolean setLightTimestamp(long LightTimestamp)
    {
        this.LightTimestamp =LightTimestamp;
        return true;
    }
    public boolean setLighhtVolume(String LighhtVolume)
    {
        this.LighhtVolume =LighhtVolume;
        return true;
    }
    public LightObject(){}
    public LightObject(String LighhtVolume, long LightTimestamp) {...}
```

Figure 34 LightObject Class

LightListener

LightListener implements the class SensorEventListener [42] which is used for receiving notifications from the SensorManager when light volume has changed. We define a DBHelper object for the database connection and SensorManager for the access of device's light sensor. When this class is called by the system, it registers the sensor manager event in order to get the values of the sensor. If the last update of the sensor is greater than 2.5 minutes (this value can change during tests) then LightListener calls the *InsertLightLogs* procedure from DBHelper object and store the values in the database.

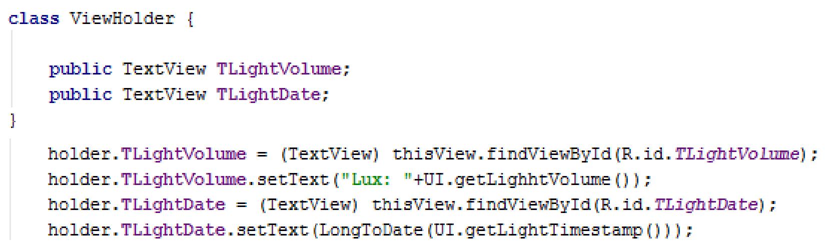


```
public class LightListener implements SensorEventListener {
    Context context;
    float lightvalue;
    SensorManager sensorManager;
    Sensor sensor;
    DBHelper dbHelper;
    private long lastUpdate = 0;

    public  LightListener(Context context)
    {
        this.context=context;
        sensorManager = (SensorManager) context.getSystemService(Context.SENSOR_SERVICE);
        sensor = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);

    @Override
        public void onSensorChanged(SensorEvent event)
        {
            dbHelper=new DBHelper(context);
            lightvalue = event.values[0];
            long curTime = System.currentTimeMillis();
            if ((curTime - lastUpdate) > 150000)//2.5min
            {

                lastUpdate = curTime;
                Log.d("Light", Float.toString(lightvalue));
                dbHelper.InsertLightLogs(Float.toString(lightvalue), lastUpdate);
            }
            dbHelper.close();
        }
```

Figure 35 LightListener - Parameters and insertion of new values

LightAdapter

LightAdapter is a class that extends an array adapter that is filled with a list of *LightObjects*. It connects the potential light data with a list view, in other words we define the look of the list and prepare the specific list in case is required by the application. This is achieved via a Viewholder class in which we define the corresponding values of the *LightObject* object and bind them with the appropriate list view that we have already created based on the *R.id* identity.

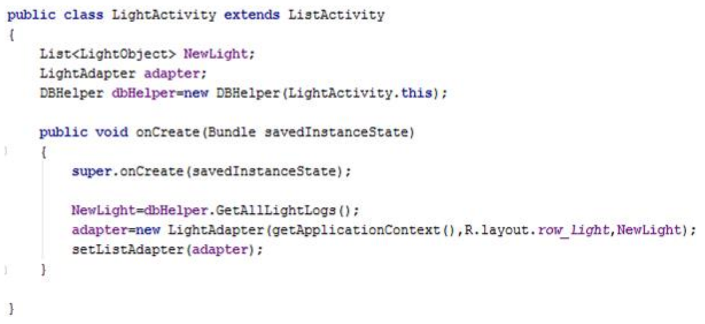


```
class ViewHolder {

    public TextView TLightVolume;
    public TextView TLightDate;
}

    holder.TLightVolume = (TextView) thisView.findViewById(R.id.TLightVolume);
    holder.TLightVolume.setText("Lux: "+UI.getLighhtVolume());
    holder.TLightDate = (TextView) thisView.findViewById(R.id.TLightDate);
    holder.TLightDate.setText(LongToDate(UI.getLightTimestamp()));
```

Figure 36 LightAdapter - Binding view and object

LightActivity

This activity is a complementary module to provide user with a visualization of data that the application is stored for later use as the previous activities. The class extends the ListActivity class and inside the class we define a DBHelper object and a list of LightObjects which are filled in with the light records from DBHelper procedure *Get-AllLightLogs.*



```
public class LightActivity extends ListActivity
{
    List<LightObject> NewLight;
    LightAdapter adapter;
    DBHelper dbHelper=new DBHelper(LightActivity.this);

    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        NewLight=dbHelper.GetAllLightLogs();
        adapter=new LightAdapter(getApplicationContext(),R.layout.row_light,NewLight);
        setListAdapter(adapter);
    }

}
```

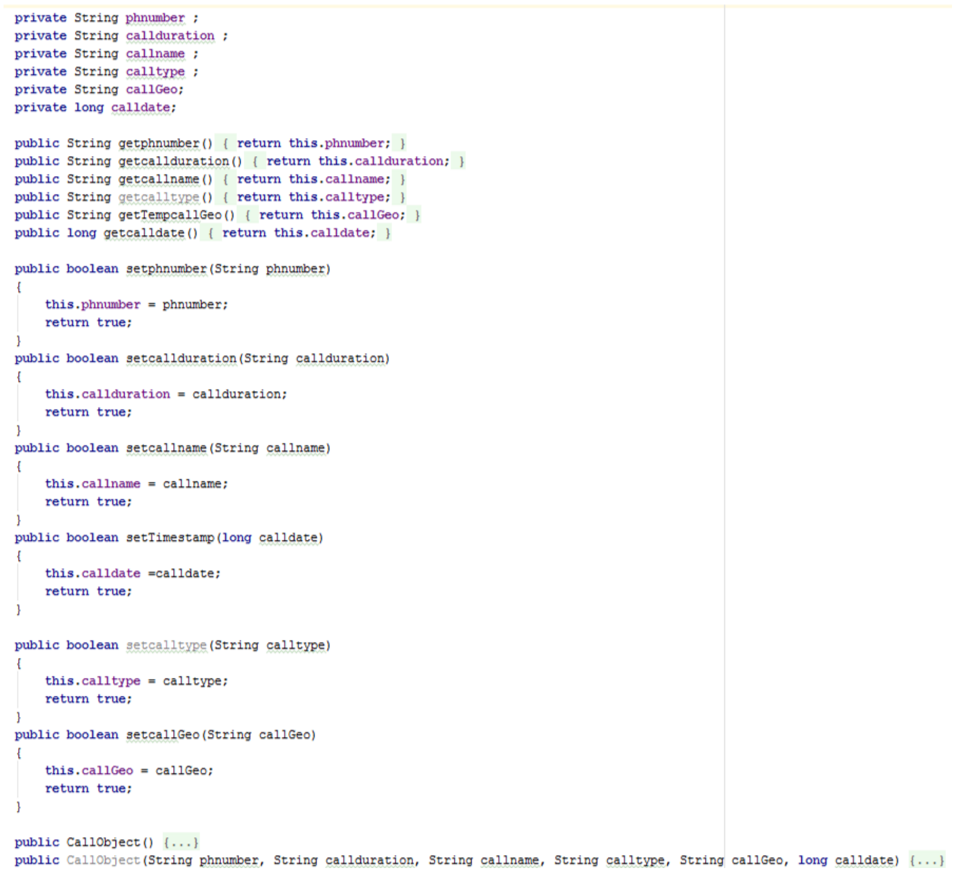
Figure 37 LightActivity - Display the available light values

4.1.5 CallLog Module

This module uses Google's CallLog [43] class that provides information about incoming and outgoing calls. We are implementing the CallLog.Calls [44] in order to fetch information regarding the call number, location, the name of the caller, the type of the call etc. Additionally we must use READ_CALL_LOG, CALL_PHONE, WRITE_CALL_LOG, READ_LOGS and READ_CONTACTS permissions to have access to device data.

CallObject

It consists of six variables of which the first mentioned in phone number; the second mentioned in the call duration in seconds; the third in name of the number holder the fourth in type of the call, the fifth in the location of the call and the last one in the timestamp. Along with the variable definition Set and Get methods are implemented as a pattern of data encapsulation. Instead of accessing class member variables directly, we define get methods to access these variables, and set methods to modify them. Finally two constructors are implemented, the default and a variation with all the proper parameters.

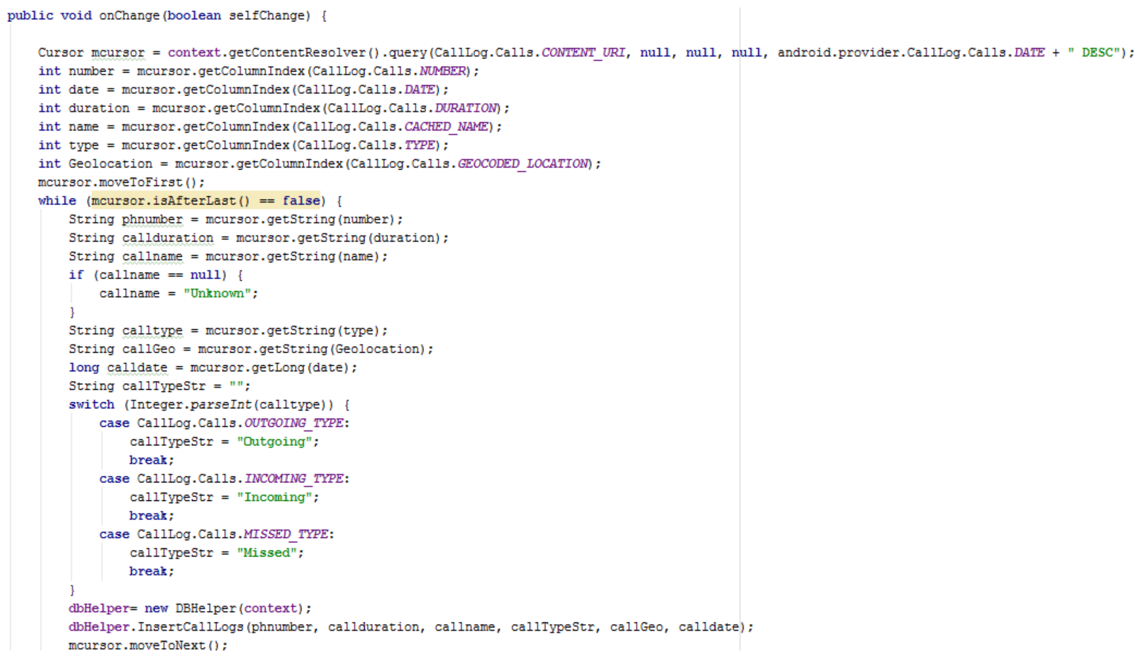


```java
private String phnumber ;
private String callduration ;
private String callname ;
private String calltype ;
private String callGeo;
private long calldate;

public String getphnumber() { return this.phnumber; }
public String getcallduration() { return this.callduration; }
public String getcallname() { return this.callname; }
public String getcalltype() { return this.calltype; }
public String getTempcallGeo() { return this.callGeo; }
public long getcalldate() { return this.calldate; }

public boolean setphnumber(String phnumber)
{
    this.phnumber = phnumber;
    return true;
}
public boolean setcallduration(String callduration)
{
    this.callduration = callduration;
    return true;
}
public boolean setcallname(String callname)
{
    this.callname = callname;
    return true;
}
public boolean setTimestamp(long calldate)
{
    this.calldate =calldate;
    return true;
}

public boolean setcalltype(String calltype)
{
    this.calltype = calltype;
    return true;
}
public boolean setcallGeo(String callGeo)
{
    this.callGeo = callGeo;
    return true;
}

public CallObject() {...}
public CallObject(String phnumber, String callduration, String callname, String calltype, String callGeo, long calldate) {...}
```

Figure 38 CallObject class

CallListener

CallListener implements the ContentObserver [45] class which is used for receiving call backs when changes to content are occurred. As in the previous listeners we define a DBHelper object for the database connection and a cursor (a form of list) for storing the information of the CallLog.Call provider. The class is called in every status change and then CallListener calls the *InsertCallLogs* procedure from DBHelper object and store the values in the database.

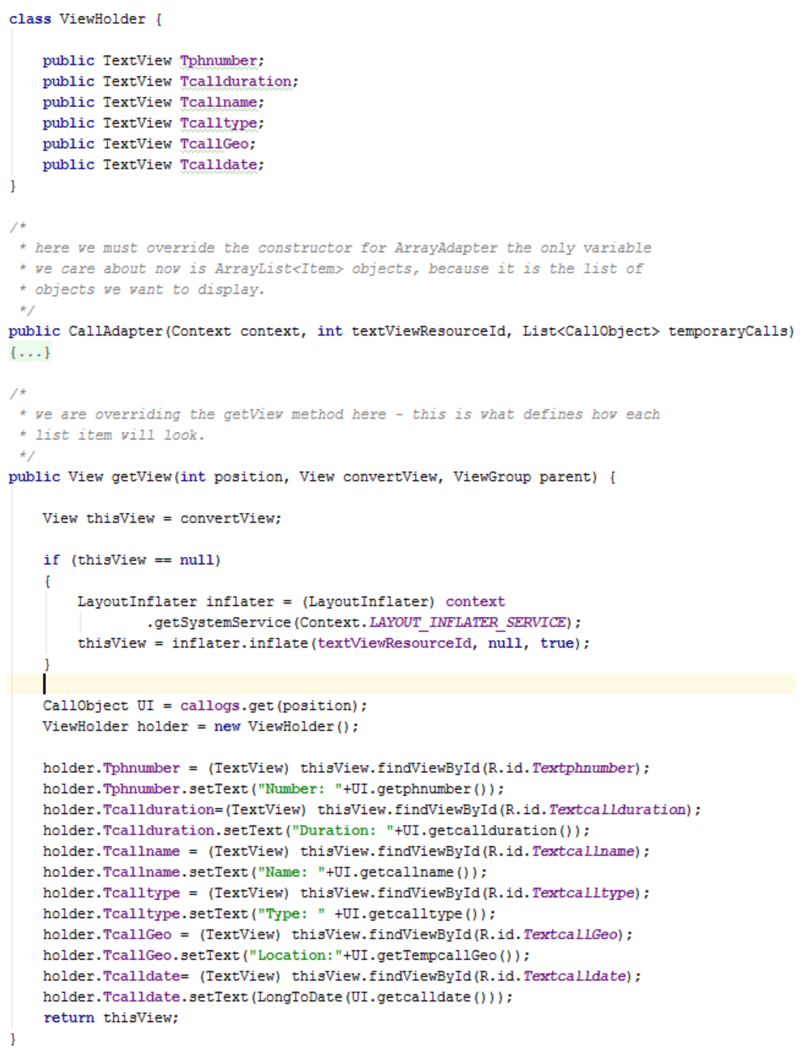


```java
public void onChange(boolean selfChange) {

    Cursor mcursor = context.getContentResolver().query(CallLog.Calls.CONTENT_URI, null, null, null, android.provider.CallLog.Calls.DATE + " DESC");
    int number = mcursor.getColumnIndex(CallLog.Calls.NUMBER);
    int date = mcursor.getColumnIndex(CallLog.Calls.DATE);
    int duration = mcursor.getColumnIndex(CallLog.Calls.DURATION);
    int name = mcursor.getColumnIndex(CallLog.Calls.CACHED_NAME);
    int type = mcursor.getColumnIndex(CallLog.Calls.TYPE);
    int Geolocation = mcursor.getColumnIndex(CallLog.Calls.GEOCODED_LOCATION);
    mcursor.moveToFirst();
    while (mcursor.isAfterLast() == false) {
        String phnumber = mcursor.getString(number);
        String callduration = mcursor.getString(duration);
        String callname = mcursor.getString(name);
        if (callname == null) {
            callname = "Unknown";
        }
        String calltype = mcursor.getString(type);
        String callGeo = mcursor.getString(Geolocation);
        long calldate = mcursor.getLong(date);
        String callTypeStr = "";
        switch (Integer.parseInt(calltype)) {
            case CallLog.Calls.OUTGOING_TYPE:
                callTypeStr = "Outgoing";
                break;
            case CallLog.Calls.INCOMING_TYPE:
                callTypeStr = "Incoming";
                break;
            case CallLog.Calls.MISSED_TYPE:
                callTypeStr = "Missed";
                break;
        }
        dbHelper= new DBHelper(context);
        dbHelper.InsertCallLogs(phnumber, callduration, callname, callTypeStr, callGeo, calldate);
        mcursor.moveToNext();
```

Figure 39 CallListener - Parameters and Insertion of new calls

CallAdapter

CallAdapter is a class that extends an array adapter that is filled with a list of *CallObjects*. It connects the potential call data with a list view, in other words we define the look of the list and prepare the specific list in case is required by the application. This is achieved via a Viewholder class in which we define the corresponding values of the *CallObject* object and bind them with the appropriate list view that we have already created based on the *R.id* identity.

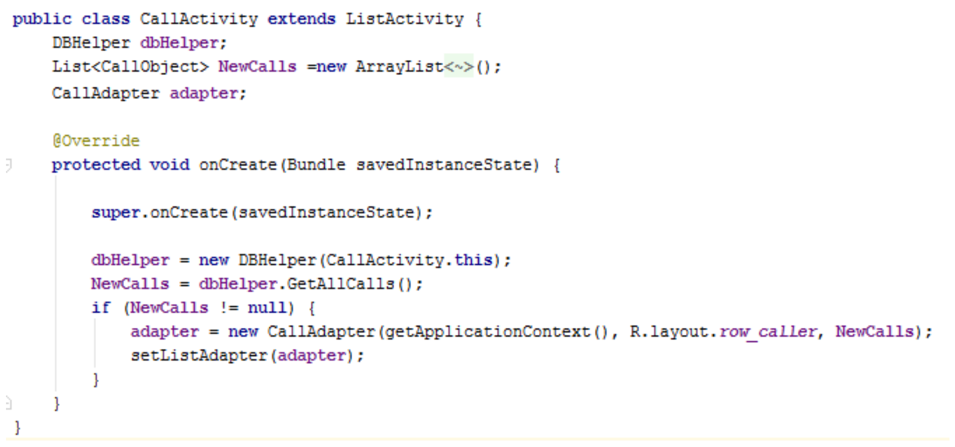


```
class ViewHolder {

    public TextView Tphnumber;
    public TextView Tcallduration;
    public TextView Tcallname;
    public TextView Tcalltype;
    public TextView TcallGeo;
    public TextView Tcalldate;
}

/*
 * here we must override the constructor for ArrayAdapter the only variable
 * we care about now is ArrayList<Item> objects, because it is the list of
 * objects we want to display.
 */
public CallAdapter(Context context, int textViewResourceId, List<CallObject> temporaryCalls)
{...}

/*
 * we are overriding the getView method here - this is what defines how each
 * list item will look.
 */
public View getView(int position, View convertView, ViewGroup parent) {

    View thisView = convertView;

    if (thisView == null)
    {
        LayoutInflater inflater = (LayoutInflater) context
                .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        thisView = inflater.inflate(textViewResourceId, null, true);
    }

    CallObject UI = callogs.get(position);
    ViewHolder holder = new ViewHolder();

    holder.Tphnumber = (TextView) thisView.findViewById(R.id.Textphnumber);
    holder.Tphnumber.setText("Number: "+UI.getphnumber());
    holder.Tcallduration=(TextView) thisView.findViewById(R.id.Textcallduration);
    holder.Tcallduration.setText("Duration: "+UI.getcallduration());
    holder.Tcallname = (TextView) thisView.findViewById(R.id.Textcallname);
    holder.Tcallname.setText("Name: "+UI.getcallname());
    holder.Tcalltype = (TextView) thisView.findViewById(R.id.Textcalltype);
    holder.Tcalltype.setText("Type: " +UI.getcalltype());
    holder.TcallGeo = (TextView) thisView.findViewById(R.id.TextcallGeo);
    holder.TcallGeo.setText("Location:"+UI.getTempcallGeo());
    holder.Tcalldate= (TextView) thisView.findViewById(R.id.Textcalldate);
    holder.Tcalldate.setText(LongToDate(UI.getcalldate()));
    return thisView;
}
```

Figure 40 CallAdapter - Binding view and object

CallActivity

Following the same pattern as the previous activities CallActivity extends the ListActivity class and inside the class we define a DBHelper object and a list of CallObjects which are filled in with the call records from DBHelper procedure *GetAllCalls.*



```java
public class CallActivity extends ListActivity {
    DBHelper dbHelper;
    List<CallObject> NewCalls =new ArrayList<~>();
    CallAdapter adapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        dbHelper = new DBHelper(CallActivity.this);
        NewCalls = dbHelper.GetAllCalls();
        if (NewCalls != null) {
            adapter = new CallAdapter(getApplicationContext(), R.layout.row_caller, NewCalls);
            setListAdapter(adapter);
        }
    }
}
```

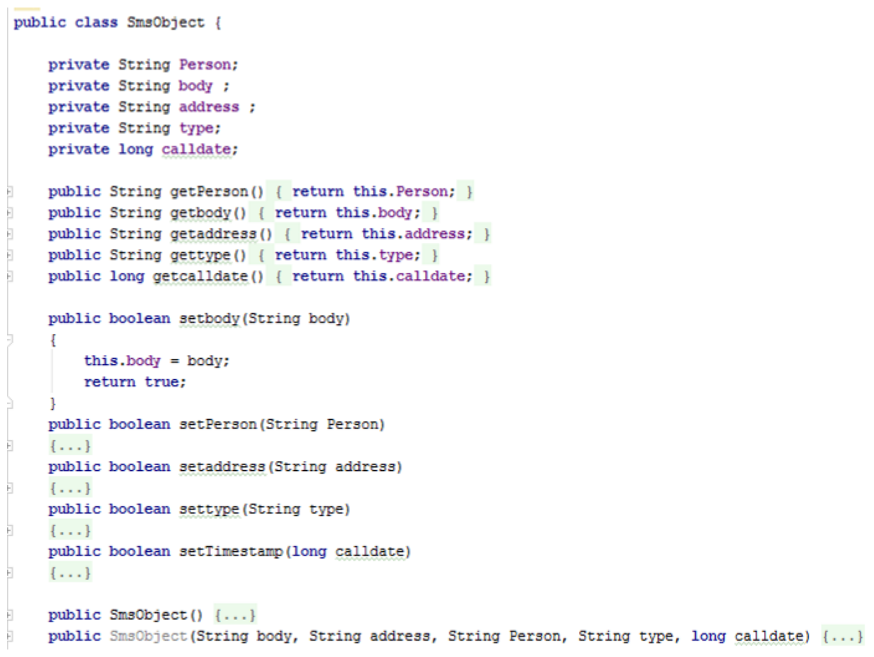
Figure 41 CallActivity - Display the available call records

4.1.6 Sms Module

This module uses Google’s Telephony.Sms [46] class that provides information about sent and received sms. We are implementing the above class in cooperation with ContactsContract [47] in order to fetch information regarding the person’s name, call number, context, type of the sms etc. Additionally we must use READ_SMS permissions to have access to device data.

SmsObject

It consists of five variables of which the first mentioned in sender’s name; the second mentioned in content of the message; the third in sender’s name; the fourth in type of the sms and the last one in the timestamp. Along with the variable definition Set and Get methods are implemented as a pattern of data encapsulation. Instead of accessing class member variables directly, we define get methods to read these variables, and set methods to modify them. Finally two constructors are implemented, the default and a variation with all the proper parameters.

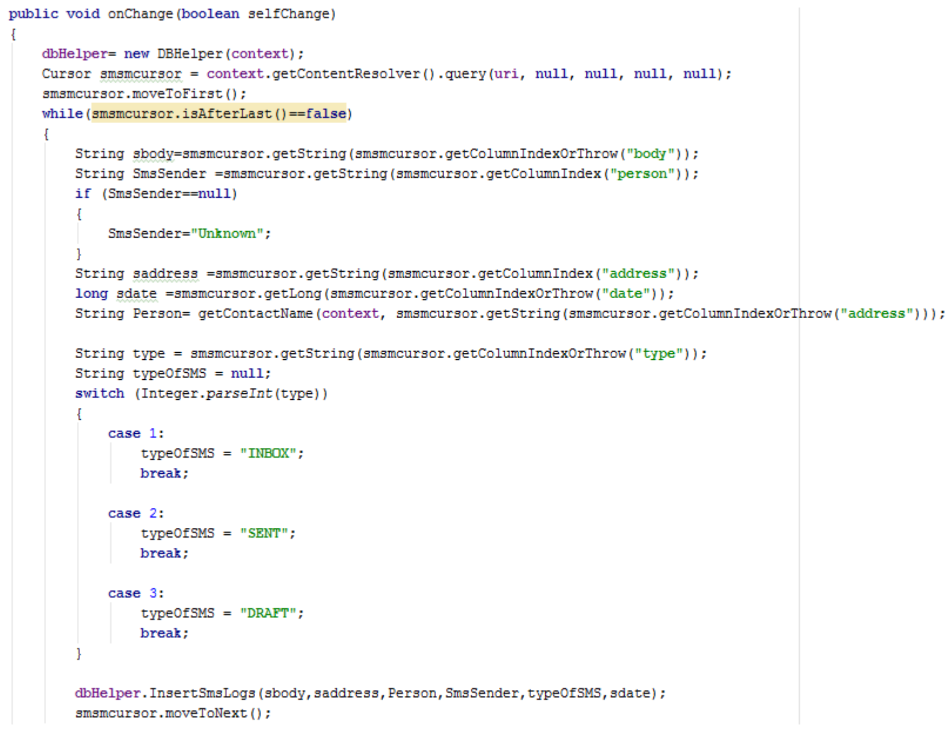


```
public class SmsObject {

    private String Person;
    private String body ;
    private String address ;
    private String type;
    private long calldate;

    public String getPerson() { return this.Person; }
    public String getbody() { return this.body; }
    public String getaddress() { return this.address; }
    public String gettype() { return this.type; }
    public long getcalldate() { return this.calldate; }

    public boolean setbody(String body)
    {
        this.body = body;
        return true;
    }
    public boolean setPerson(String Person)
    {...}
    public boolean setaddress(String address)
    {...}
    public boolean settype(String type)
    {...}
    public boolean setTimestamp(long calldate)
    {...}

    public SmsObject() {...}
    public SmsObject(String body, String address, String Person, String type, long calldate) {...}
```

Figure 42 SmsObject Class - Display the available call records

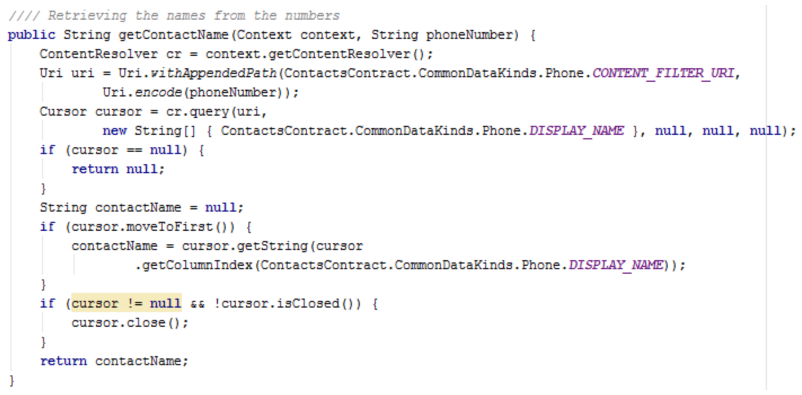
SmsListener

SmsListener implements the ContentObserver [45] class which is used for receiving call backs when changes to content are occurred. As in the previous listeners we define a DBHelper object for the database connection and a cursor (a form of list) for storing the information of the sms provider. The class is called in every status change and then SmsListener calls the *getContactName* for matching the number and the name of the sender and afterwards calls the *InsertSmsLogs* procedure from DBHelper object and store the values in the database.



```java
public void onChange(boolean selfChange)
{
    dbHelper= new DBHelper(context);
    Cursor smsmcursor = context.getContentResolver().query(uri, null, null, null, null);
    smsmcursor.moveToFirst();
    while(smsmcursor.isAfterLast()==false)
    {
        String sbody=smsmcursor.getString(smsmcursor.getColumnIndexOrThrow("body"));
        String SmsSender =smsmcursor.getString(smsmcursor.getColumnIndex("person"));
        if (SmsSender==null)
        {
            SmsSender="Unknown";
        }
        String saddress =smsmcursor.getString(smsmcursor.getColumnIndex("address"));
        long sdate =smsmcursor.getLong(smsmcursor.getColumnIndexOrThrow("date"));
        String Person= getContactName(context, smsmcursor.getString(smsmcursor.getColumnIndexOrThrow("address")));

        String type = smsmcursor.getString(smsmcursor.getColumnIndexOrThrow("type"));
        String typeOfSMS = null;
        switch (Integer.parseInt(type))
        {
            case 1:
                typeOfSMS = "INBOX";
                break;

            case 2:
                typeOfSMS = "SENT";
                break;

            case 3:
                typeOfSMS = "DRAFT";
                break;
        }

        dbHelper.InsertSmsLogs(sbody,saddress,Person,SmsSender,typeOfSMS,sdate);
        smsmcursor.moveToNext();
```

Figure 43 SmsListener - Parameters and Insertion of new sms (1)

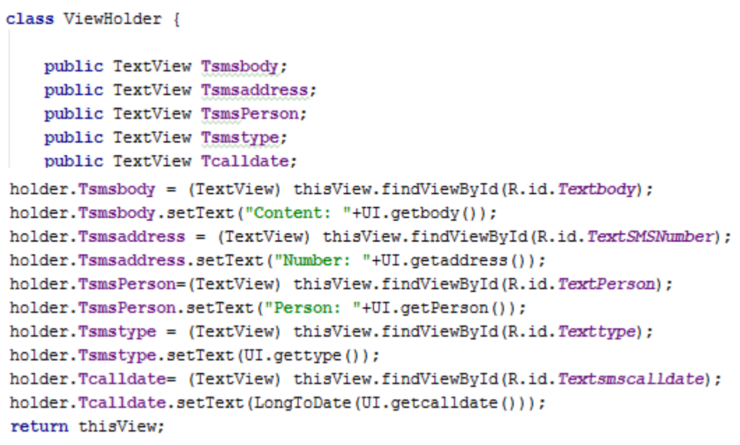


```java
//// Retrieving the names from the numbers
public String getContactName(Context context, String phoneNumber) {
    ContentResolver cr = context.getContentResolver();
    Uri uri = Uri.withAppendedPath(ContactsContract.CommonDataKinds.Phone.CONTENT_FILTER_URI,
            Uri.encode(phoneNumber));
    Cursor cursor = cr.query(uri,
            new String[] { ContactsContract.CommonDataKinds.Phone.DISPLAY_NAME }, null, null, null);
    if (cursor == null) {
        return null;
    }
    String contactName = null;
    if (cursor.moveToFirst()) {
        contactName = cursor.getString(cursor
                .getColumnIndex(ContactsContract.CommonDataKinds.Phone.DISPLAY_NAME));
    }
    if (cursor != null && !cursor.isClosed()) {
        cursor.close();
    }
    return contactName;
}
```

Figure 44 SmsListener - Parameters and Insertion of new sms (2)

SmsAdapter

SmsAdapter is a class that extends an array adapter that is filled with a list of *SmsObjects*. It connects the potential sms data with a list view, in other words we define the look of the list and prepare the specific list in case is required by the application. This is achieved via a Viewholder class in which we define the corresponding values of the *SmsObject* object and bind them with the appropriate list view that we have already created based on the *R.id* identity

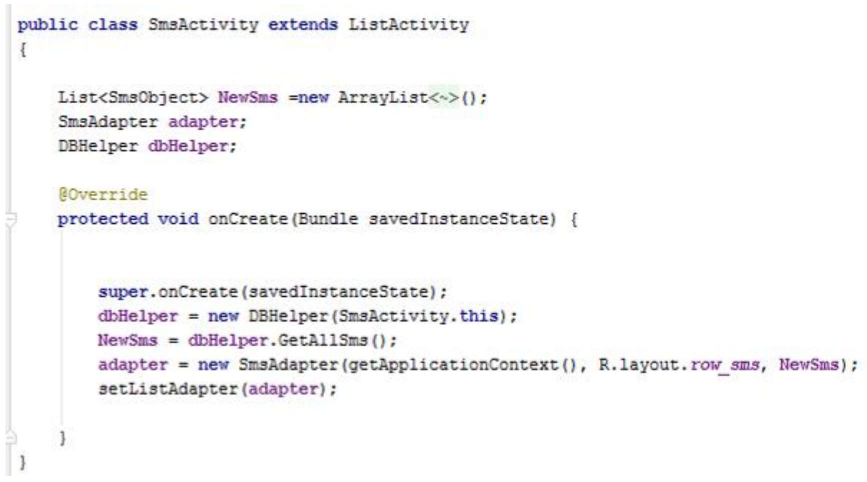


```
class ViewHolder {

    public TextView Tsmsbody;
    public TextView Tsmsaddress;
    public TextView TsmsPerson;
    public TextView Tsmstype;
    public TextView Tcalldate;
holder.Tsmsbody = (TextView) thisView.findViewById(R.id.Textbody);
holder.Tsmsbody.setText("Content: "+UI.getbody());
holder.Tsmsaddress = (TextView) thisView.findViewById(R.id.TextSMSNumber);
holder.Tsmsaddress.setText("Number: "+UI.getaddress());
holder.TsmsPerson=(TextView) thisView.findViewById(R.id.TextPerson);
holder.TsmsPerson.setText("Person: "+UI.getPerson());
holder.Tsmstype = (TextView) thisView.findViewById(R.id.Texttype);
holder.Tsmstype.setText(UI.gettype());
holder.Tcalldate= (TextView) thisView.findViewById(R.id.Textsmscalldate);
holder.Tcalldate.setText(LongToDate(UI.getcalldate()));
return thisView;
```

Figure 45 SmsAdapter - Binding view and object

SmsActivity

As the aforementioned activities, CallActivity extends the ListActivity class and inside the class we define a DBHelper object and a list of SmsObjects which are filled in with the sms records from DBHelper procedure *GetAllSms.*



```
public class SmsActivity extends ListActivity
{

    List<SmsObject> NewSms =new ArrayList<~>();
    SmsAdapter adapter;
    DBHelper dbHelper;

    @Override
    protected void onCreate(Bundle savedInstanceState) {


        super.onCreate(savedInstanceState);
        dbHelper = new DBHelper(SmsActivity.this);
        NewSms = dbHelper.GetAllSms();
        adapter = new SmsAdapter(getApplicationContext(), R.layout.row_sms, NewSms);
        setListAdapter(adapter);

    }
}
```

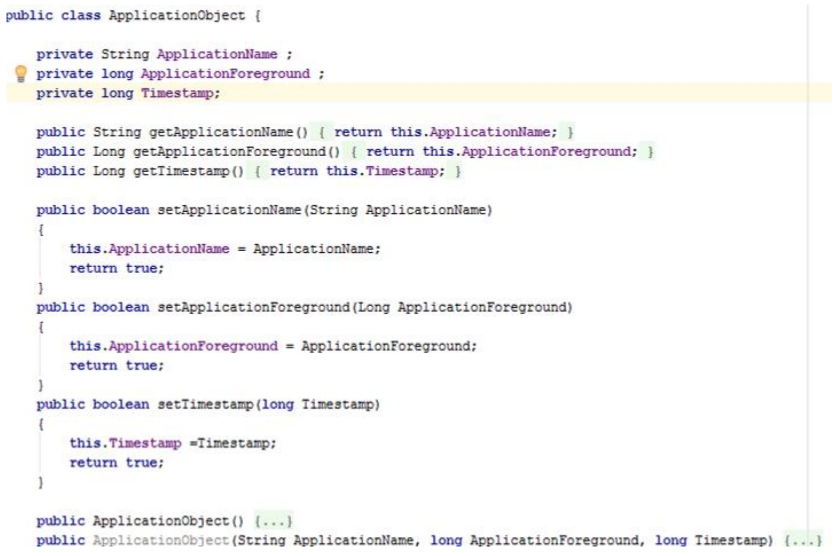
Figure 46 SmsActivity - Display the available sms records

4.1.7 Application Module

This module uses Google's PackageManager [48] class that provides information about the application packages that are currently installed on the device. We are implementing the above class in cooperation with UsageStatManager [49] in order to fetch information regarding the application statistics. We must use ACTION_USAGE_ACCESS_SETTINGS or GET_TASKS permissions to have access to device data.

ApplicationObject

It consists of three variables of which the first mentioned in application’s name; the second mentioned in foreground duration in milliseconds and the third one in the timestamp. Along with the variable definition Set and Get methods are implemented as a pattern of data encapsulation. Instead of accessing class member variables directly, we define get methods to access these variables, and set methods to modify them. Finally two constructors are implemented, the default and a variation with all the proper parameters.



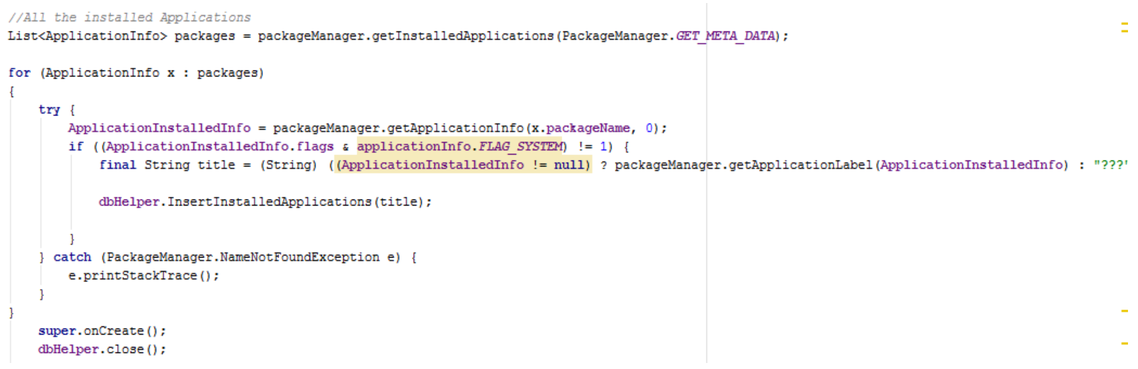
```
public class ApplicationObject {

    private String ApplicationName ;
    private long ApplicationForeground ;
    private long Timestamp;

    public String getApplicationName() { return this.ApplicationName; }
    public Long getApplicationForeground() { return this.ApplicationForeground; }
    public Long getTimestamp() { return this.Timestamp; }

    public boolean setApplicationName(String ApplicationName)
    {
        this.ApplicationName = ApplicationName;
        return true;
    }
    public boolean setApplicationForeground(Long ApplicationForeground)
    {
        this.ApplicationForeground = ApplicationForeground;
        return true;
    }
    public boolean setTimestamp(long Timestamp)
    {
        this.Timestamp =Timestamp;
        return true;
    }

    public ApplicationObject() {...}
    public ApplicationObject(String ApplicationName, long ApplicationForeground, long Timestamp) {...}
```

Figure 47 ApplicationObject Class

ApplicationListener

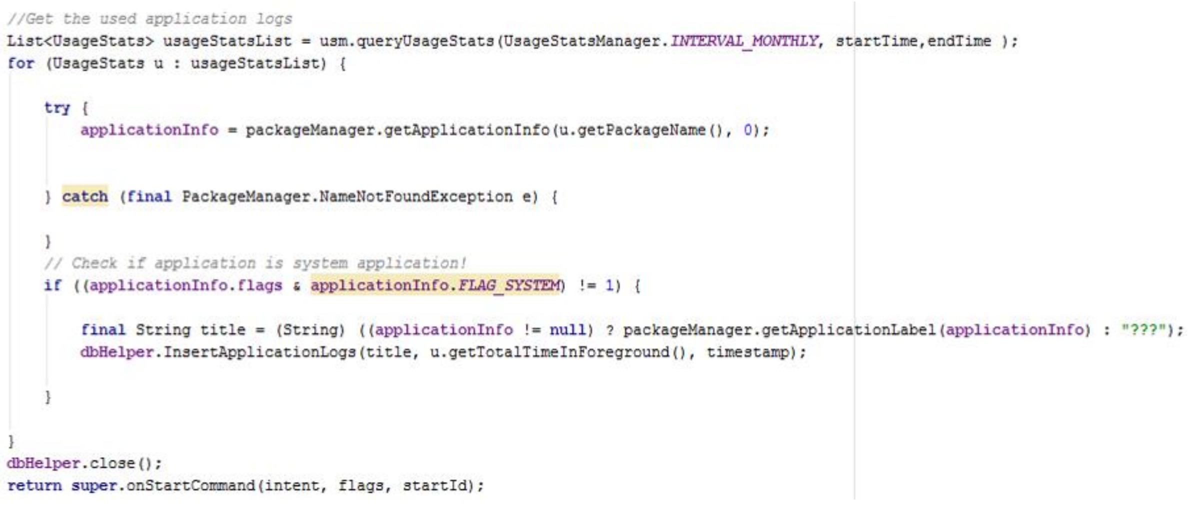
The ApplicationListener is an extension of a Service [50] and we implemented two distinct logics. The first relates to installed applications in which we call the PackageManager on the creation of the service in order to fetch from the system the list of the installed applications and afterwards to store the list in the database with the DBHelper procedure *InsertInstalledApplications.* The second one relates to the application logger in which we call again the PackageManager with specific parameters in order to get the monthly foreground duration log, excluding the system’s applications, and store it to the database with DBHelper’s *InsertApplicationLogs* procedure.

The month duration is calculated based on the current date.



```
//All the installed Applications
List<ApplicationInfo> packages = packageManager.getInstalledApplications(PackageManager.GET_META_DATA);

for (ApplicationInfo x : packages)
{
    try {
        ApplicationInstalledInfo = packageManager.getApplicationInfo(x.packageName, 0);
        if ((ApplicationInstalledInfo.flags & applicationInfo.FLAG_SYSTEM) != 1) {
            final String title = (String) ((ApplicationInstalledInfo != null) ? packageManager.getApplicationLabel(ApplicationInstalledInfo) : "???'

            dbHelper.InsertInstalledApplications(title);

        }
    } catch (PackageManager.NameNotFoundException e) {
        e.printStackTrace();
    }
}
    super.onCreate();
    dbHelper.close();
```

Figure 48 ApplicationListener – All installed applications

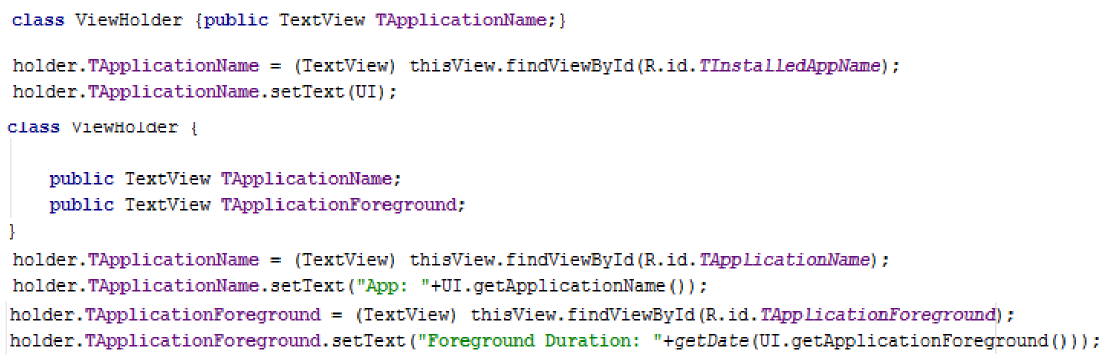


```
//Get the used application logs
List<UsageStats> usageStatsList = usm.queryUsageStats(UsageStatsManager.INTERVAL_MONTHLY, startTime,endTime );
for (UsageStats u : usageStatsList) {

    try {
        applicationInfo = packageManager.getApplicationInfo(u.getPackageName(), 0);


    } catch (final PackageManager.NameNotFoundException e) {

    }
    // Check if application is system application!
    if ((applicationInfo.flags & applicationInfo.FLAG_SYSTEM) != 1) {

        final String title = (String) ((applicationInfo != null) ? packageManager.getApplicationLabel(applicationInfo) : "???");
        dbHelper.InsertApplicationLogs(title, u.getTotalTimeInForeground(), timestamp);

    }

}
dbHelper.close();
return super.onStartCommand(intent, flags, startId);
```

Figure 49 ApplicationListener – Application log

ApplicationAdapter

ApplicationAdapter is a class that extends an array adapter that is filled with a list of *ApplicationObjects*. Inside the Viewholder class we define the corresponding values of the *ApplicationObject* object and bind them with the appropriate list view that we have already created based on the *R.id* identity. In this case we created two different adapters one for the installed applications and one for interacted applications.

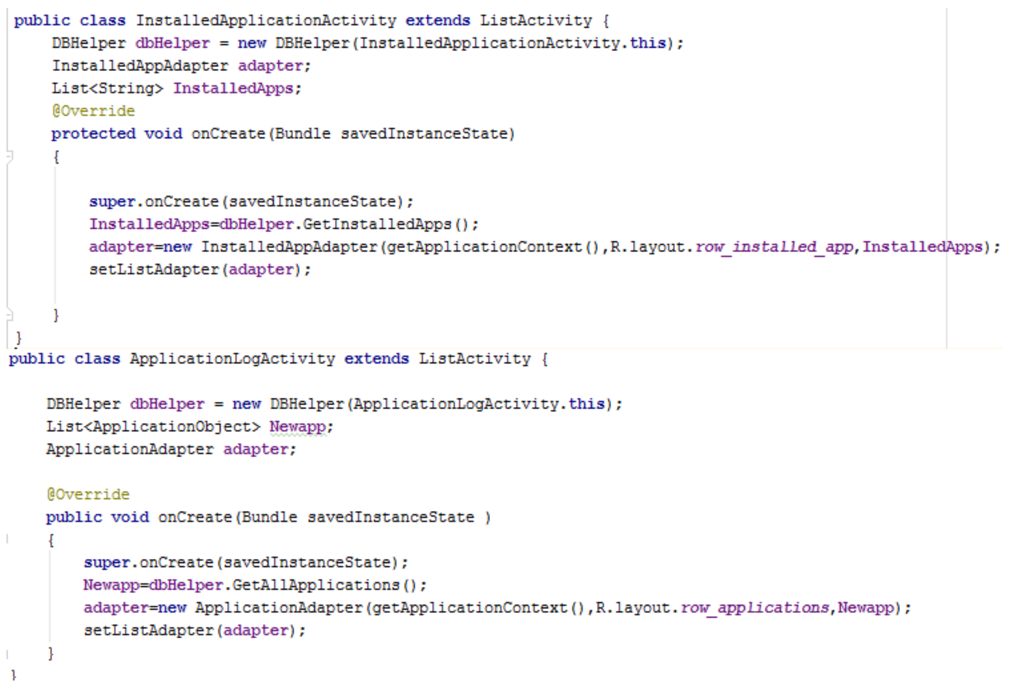


```
class ViewHolder {public TextView TApplicationName;}

holder.TApplicationName = (TextView) thisView.findViewById(R.id.TInstalledAppName);
holder.TApplicationName.setText(UI);
class ViewHolder {

    public TextView TApplicationName;
    public TextView TApplicationForeground;
}
holder.TApplicationName = (TextView) thisView.findViewById(R.id.TApplicationName);
holder.TApplicationName.setText("App: "+UI.getApplicationName());
holder.TApplicationForeground = (TextView) thisView.findViewById(R.id.TApplicationForeground);
holder.TApplicationForeground.setText("Foreground Duration: "+getDate(UI.getApplicationForeground()));
```

Figure 50 ApplicationAdapter -Binding views with objects

ApplicationActivity

In this case we implement two different activities one (InstalledApplicationActivity) for displaying all the installed applications of the mobile device and one (Application-LogActivity) displaying the applications that the user interacts with. Both extend the ListActivity class and inside the class we define a DBHelper object and a list of Appli-cationObject which are filled in with the data records from DBHelper procedure *Get-InstalledApps* and *GetAllApplications* respectively.



```
public class InstalledApplicationActivity extends ListActivity {
    DBHelper dbHelper = new DBHelper(InstalledApplicationActivity.this);
    InstalledAppAdapter adapter;
    List<String> InstalledApps;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {

        super.onCreate(savedInstanceState);
        InstalledApps=dbHelper.GetInstalledApps();
        adapter=new InstalledAppAdapter(getApplicationContext(),R.layout.row_installed_app,InstalledApps);
        setListAdapter(adapter);

    }
}
public class ApplicationLogActivity extends ListActivity {

    DBHelper dbHelper = new DBHelper(ApplicationLogActivity.this);
    List<ApplicationObject> Newapp;
    ApplicationAdapter adapter;

    @Override
    public void onCreate(Bundle savedInstanceState )
    {
        super.onCreate(savedInstanceState);
        Newapp=dbHelper.GetAllApplications();
        adapter=new ApplicationAdapter(getApplicationContext(),R.layout.row_applications,Newapp);
        setListAdapter(adapter);
    }
}
```

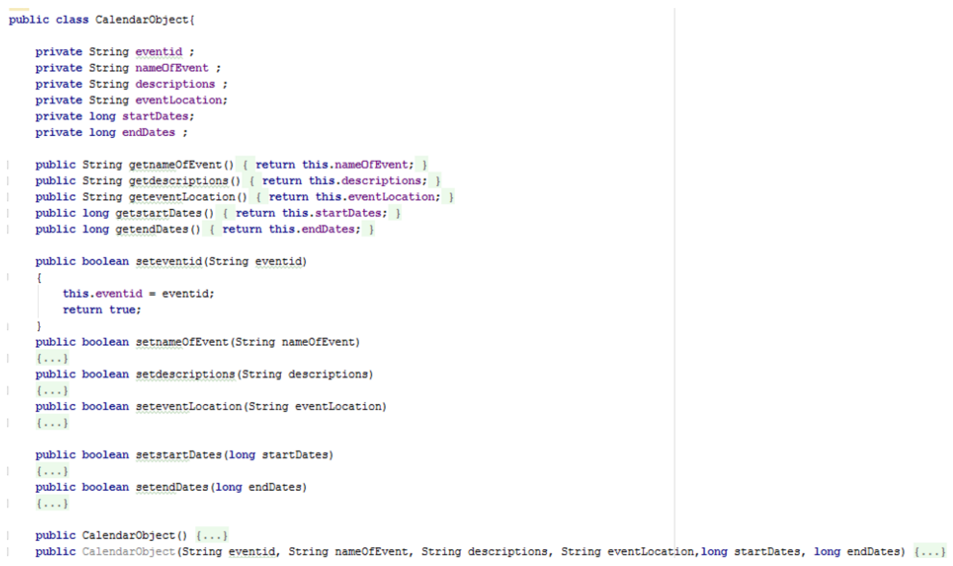
Figure 51 ApplicationActivity – Display the available app records (Both cases)

4.1.8 Calendar Module

This module uses Google’s Calendar Provider [51] class that provides information about user's calendar events. We are implementing the above class only to Google calendar with *READ_CALENDAR* permissions to have access to device in order to fetch information regarding the calendar events.

CalendarObject

It consists of six variables of which the first mentioned in event id; the second mentioned in event's name and the third one in event's description, the forth one in event's location, the fifth one in event's start date and the last in event's end date. Along with the variable definition Set and Get methods are implemented as a pattern of data encapsulation. Instead of accessing class member variables directly, we define get methods to access these variables, and set methods to modify them. Finally two constructors are implemented, the default and a variation with all the proper parameters.

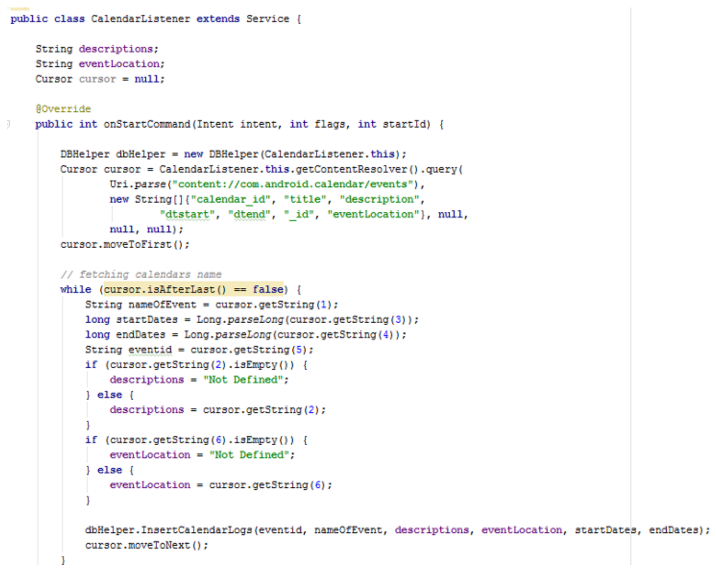


```
public class CalendarObject{

    private String eventid ;
    private String nameOfEvent ;
    private String descriptions ;
    private String eventLocation;
    private long startDates;
    private long endDates ;

    public String getnameOfEvent() { return this.nameOfEvent; }
    public String getdescriptions() { return this.descriptions; }
    public String geteventLocation() { return this.eventLocation; }
    public long getstartDates() { return this.startDates; }
    public long getendDates() { return this.endDates; }

    public boolean seteventid(String eventid)
    {
        this.eventid = eventid;
        return true;
    }
    public boolean setnameOfEvent(String nameOfEvent)
    {...}
    public boolean setdescriptions(String descriptions)
    {...}
    public boolean seteventLocation(String eventLocation)
    {...}

    public boolean setstartDates(long startDates)
    {...}
    public boolean setendDates(long endDates)
    {...}

    public CalendarObject() {...}
    public CalendarObject(String eventid, String nameOfEvent, String descriptions, String eventLocation,long startDates, long endDates) {...}
```

Figure 52 CalendarObject class

CalendarListener

The CalendarListener is an extension of a Service [50] that fetches and stores calendar events for the *content://com.android.calendar/events* into a cursor. When this procedure is over it calls DBHelper *InsertCalendarLogs* in order to store the characteristics of an event in the database.

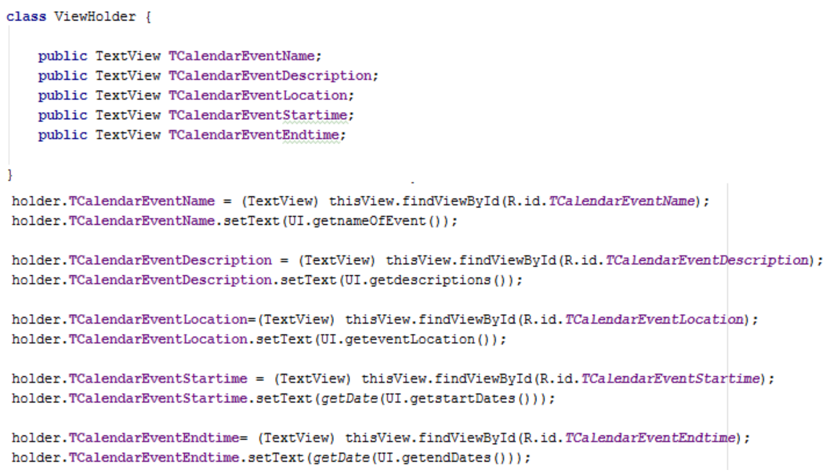


```
public class CalendarListener extends Service {

    String descriptions;
    String eventLocation;
    Cursor cursor = null;

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {

        DBHelper dbHelper = new DBHelper(CalendarListener.this);
        Cursor cursor = CalendarListener.this.getContentResolver().query(
                Uri.parse("content://com.android.calendar/events"),
                new String[]{"calendar_id", "title", "description",
                        "dtstart", "dtend", "_id", "eventLocation"}, null,
                null, null);
        cursor.moveToFirst();

        // fetching calendars name
        while (cursor.isAfterLast() == false) {
            String nameOfEvent = cursor.getString(1);
            long startDates = Long.parseLong(cursor.getString(3));
            long endDates = Long.parseLong(cursor.getString(4));
            String eventid = cursor.getString(5);
            if (cursor.getString(2).isEmpty()) {
                descriptions = "Not Defined";
            } else {
                descriptions = cursor.getString(2);
            }
            if (cursor.getString(6).isEmpty()) {
                eventLocation = "Not Defined";
            } else {
                eventLocation = cursor.getString(6);
            }

            dbHelper.InsertCalendarLogs(eventid, nameOfEvent, descriptions, eventLocation, startDates, endDates);
            cursor.moveToNext();
        }
```

Figure 53 CalendarListener - Parameters and Insertion of new events (2)

CalendarAdapter

CalendarAdapter is a class that extends an array adapter that is filled with a list of *CalendarObjects*. Inside the Viewholder class we define the corresponding values of the *CalendarObject* object and bind them with the appropriate list view that we have already created based on the *R.id* identity.

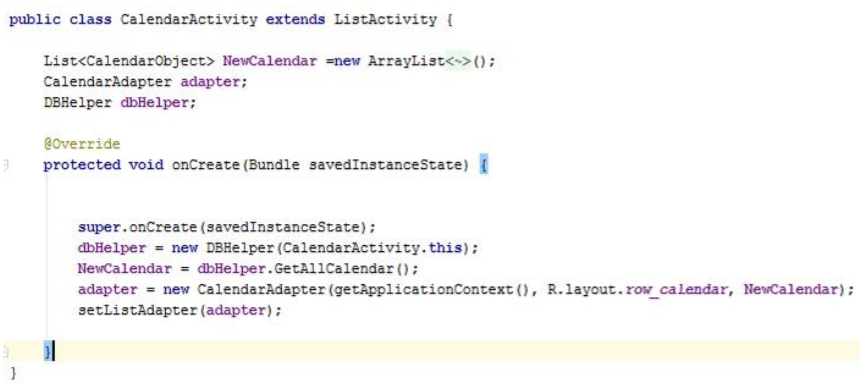


```
class ViewHolder {

    public TextView TCalendarEventName;
    public TextView TCalendarEventDescription;
    public TextView TCalendarEventLocation;
    public TextView TCalendarEventStartime;
    public TextView TCalendarEventEndtime;

}
holder.TCalendarEventName = (TextView) thisView.findViewById(R.id.TCalendarEventName);
holder.TCalendarEventName.setText(UI.getnameOfEvent());

holder.TCalendarEventDescription = (TextView) thisView.findViewById(R.id.TCalendarEventDescription);
holder.TCalendarEventDescription.setText(UI.getdescriptions());

holder.TCalendarEventLocation=(TextView) thisView.findViewById(R.id.TCalendarEventLocation);
holder.TCalendarEventLocation.setText(UI.geteventLocation());

holder.TCalendarEventStartime = (TextView) thisView.findViewById(R.id.TCalendarEventStartime);
holder.TCalendarEventStartime.setText(getDate(UI.getstartDates()));

holder.TCalendarEventEndtime= (TextView) thisView.findViewById(R.id.TCalendarEventEndtime);
holder.TCalendarEventEndtime.setText(getDate(UI.getendDates()));
```

Figure 54 CalendarAdapter -Binding views with objects

CalendarActivity

As the previous activities, CalendarActivity extends the ListActivity class and inside the class we define a DBHelper object and a list of CalendarObjects which are filled in with the calendar records from DBHelper procedure *GetAllCalendar*.

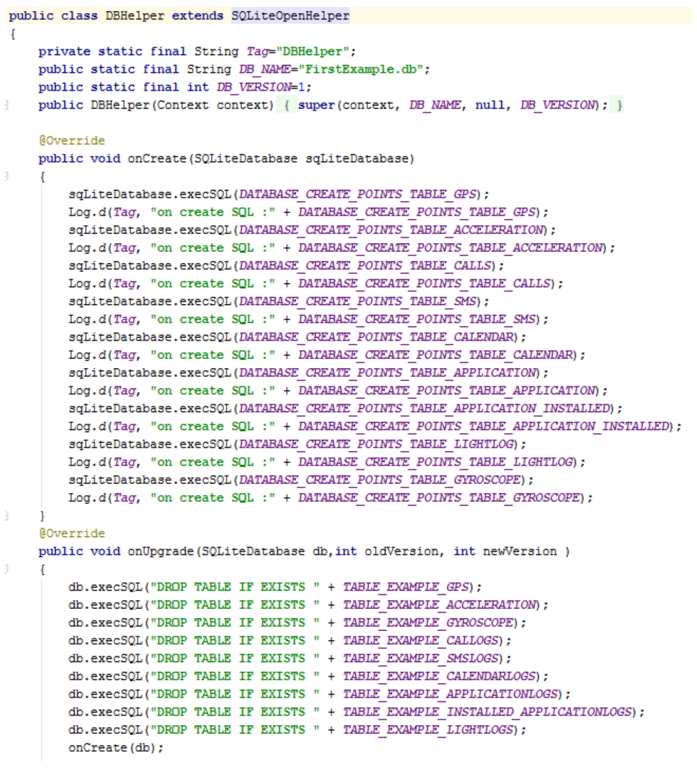


```java
public class CalendarActivity extends ListActivity {

    List<CalendarObject> NewCalendar =new ArrayList<~>();
    CalendarAdapter adapter;
    DBHelper dbHelper;

    @Override
    protected void onCreate(Bundle savedInstanceState) {


        super.onCreate(savedInstanceState);
        dbHelper = new DBHelper(CalendarActivity.this);
        NewCalendar = dbHelper.GetAllCalendar();
        adapter = new CalendarAdapter(getApplicationContext(), R.layout.row_calendar, NewCalendar);
        setListAdapter(adapter);

    }
}
```

Figure 55 CalendarActivity – Display the available calendar events

4.1.9 DBHelper Module

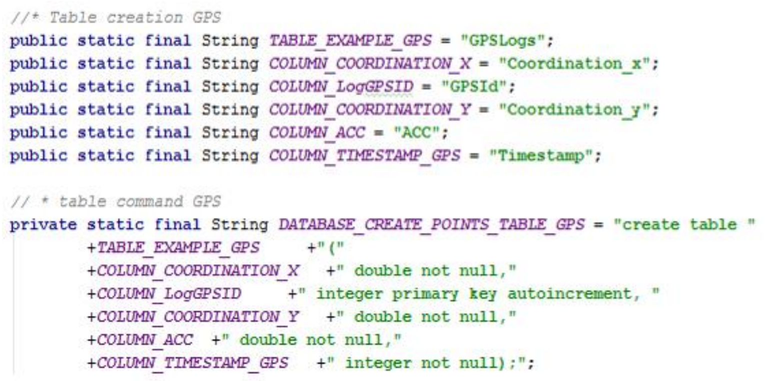
This Module is responsible for ensuring communication of the aforementioned modules with application's database. It initializes the database, creates the necessary tables for the application as well as the procedures that are mandatory for the communication with other modules. Each module has its own specific procedures for data registration as also for fetching data from the database in order to display the corresponding information to mobile user. In case needed we can enhance the database with additional components due to the modular architecture we have implemented. The first steps we execute, are the initialization of the database along with the creation of the desired tables.



```java
public class DBHelper extends SQLiteOpenHelper
{
    private static final String Tag="DBHelper";
    public static final String DB_NAME="FirstExample.db";
    public static final int DB_VERSION=1;
    public DBHelper(Context context) { super(context, DB_NAME, null, DB_VERSION); }

    @Override
    public void onCreate(SQLiteDatabase sqLiteDatabase)
    {
        sqLiteDatabase.execSQL(DATABASE_CREATE_POINTS_TABLE_GPS);
        Log.d(Tag, "on create SQL :" + DATABASE_CREATE_POINTS_TABLE_GPS);
        sqLiteDatabase.execSQL(DATABASE_CREATE_POINTS_TABLE_ACCELERATION);
        Log.d(Tag, "on create SQL :" + DATABASE_CREATE_POINTS_TABLE_ACCELERATION);
        sqLiteDatabase.execSQL(DATABASE_CREATE_POINTS_TABLE_CALLS);
        Log.d(Tag, "on create SQL :" + DATABASE_CREATE_POINTS_TABLE_CALLS);
        sqLiteDatabase.execSQL(DATABASE_CREATE_POINTS_TABLE_SMS);
        Log.d(Tag, "on create SQL :" + DATABASE_CREATE_POINTS_TABLE_SMS);
        sqLiteDatabase.execSQL(DATABASE_CREATE_POINTS_TABLE_CALENDAR);
        Log.d(Tag, "on create SQL :" + DATABASE_CREATE_POINTS_TABLE_CALENDAR);
        sqLiteDatabase.execSQL(DATABASE_CREATE_POINTS_TABLE_APPLICATION);
        Log.d(Tag, "on create SQL :" + DATABASE_CREATE_POINTS_TABLE_APPLICATION);
        sqLiteDatabase.execSQL(DATABASE_CREATE_POINTS_TABLE_APPLICATION_INSTALLED);
        Log.d(Tag, "on create SQL :" + DATABASE_CREATE_POINTS_TABLE_APPLICATION_INSTALLED);
        sqLiteDatabase.execSQL(DATABASE_CREATE_POINTS_TABLE_LIGHTLOG);
        Log.d(Tag, "on create SQL :" + DATABASE_CREATE_POINTS_TABLE_LIGHTLOG);
        sqLiteDatabase.execSQL(DATABASE_CREATE_POINTS_TABLE_GYROSCOPE);
        Log.d(Tag, "on create SQL :" + DATABASE_CREATE_POINTS_TABLE_GYROSCOPE);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db,int oldVersion, int newVersion )
    {
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_EXAMPLE_GPS);
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_EXAMPLE_ACCELERATION);
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_EXAMPLE_GYROSCOPE);
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_EXAMPLE_CALLOGS);
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_EXAMPLE_SMSLOGS);
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_EXAMPLE_CALENDARLOGS);
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_EXAMPLE_APPLICATIONLOGS);
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_EXAMPLE_INSTALLED_APPLICATIONLOGS);
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_EXAMPLE_LIGHTLOGS);
        onCreate(db);
```

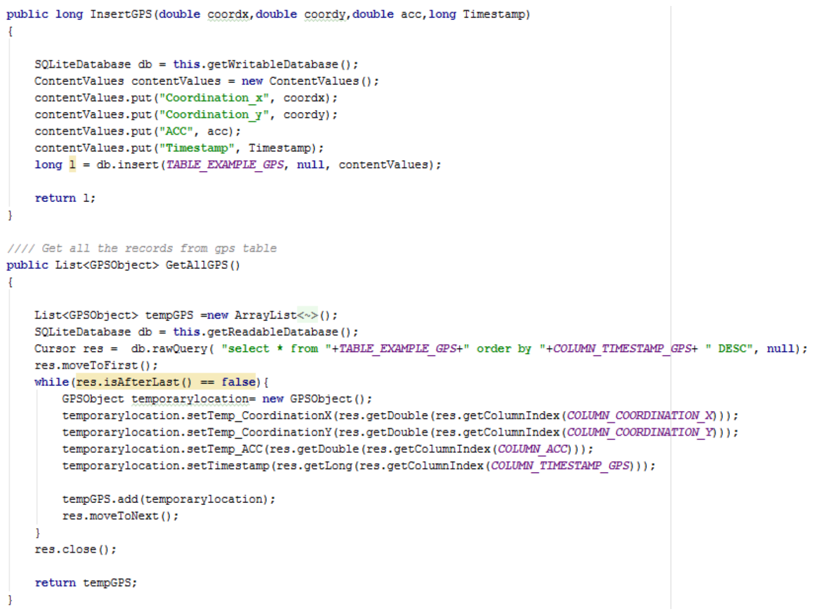
Figure 56 BDHelper - Database Initialization

We continue with the initialization of each table separately along with the stored procedures respectively. For brevity of speech we will display only the Location module table and procedures, the rest of the modules follow the same pattern.



```java
//* Table creation GPS
public static final String TABLE_EXAMPLE_GPS = "GPSLogs";
public static final String COLUMN_COORDINATION_X = "Coordination_x";
public static final String COLUMN_LogGPSID = "GPSId";
public static final String COLUMN_COORDINATION_Y = "Coordination_y";
public static final String COLUMN_ACC = "ACC";
public static final String COLUMN_TIMESTAMP_GPS = "Timestamp";

// * table command GPS
private static final String DATABASE_CREATE_POINTS_TABLE_GPS = "create table "
        +TABLE_EXAMPLE_GPS      +"("
        +COLUMN_COORDINATION_X   +" double not null,"
        +COLUMN_LogGPSID      +" integer primary key autoincrement, "
        +COLUMN_COORDINATION_Y   +" double not null,"
        +COLUMN_ACC  +" double not null,"
        +COLUMN_TIMESTAMP_GPS   +" integer not null);";
```

Figure 57 DBHelper – Location table

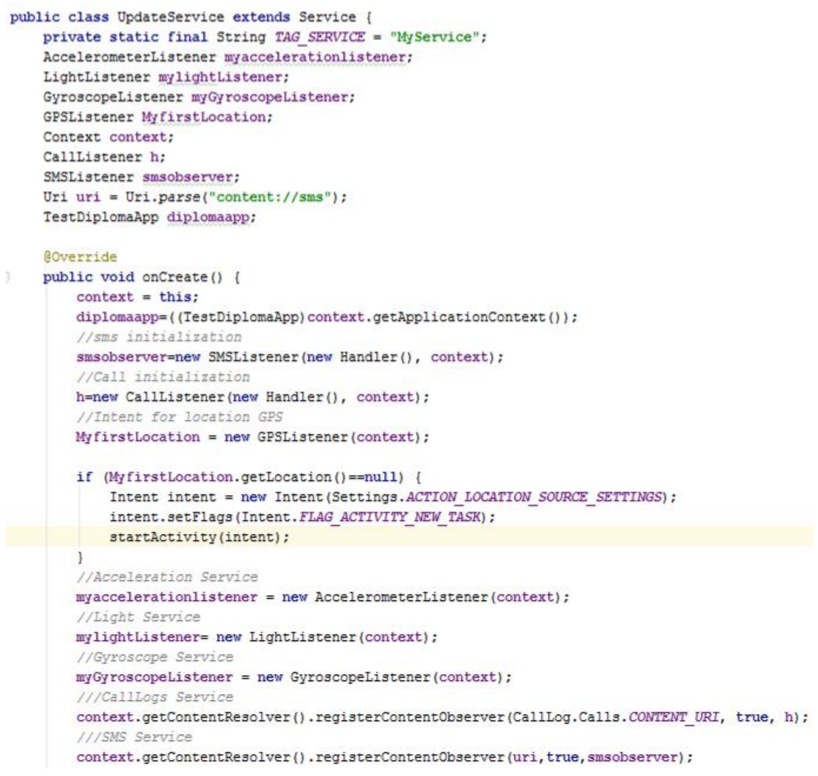


```java
public long InsertGPS(double coordx,double coordy,double acc,long Timestamp)
{

    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues contentValues = new ContentValues();
    contentValues.put("Coordination_x", coordx);
    contentValues.put("Coordination_y", coordy);
    contentValues.put("ACC", acc);
    contentValues.put("Timestamp", Timestamp);
    long l = db.insert(TABLE_EXAMPLE_GPS, null, contentValues);

    return l;
}

//// Get all the records from gps table
public List<GPSObject> GetAllGPS()
{

    List<GPSObject> tempGPS =new ArrayList<~>();
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor res =  db.rawQuery( "select * from "+TABLE_EXAMPLE_GPS+" order by "+COLUMN_TIMESTAMP_GPS+ " DESC", null);
    res.moveToFirst();
    while(res.isAfterLast() == false){
        GPSObject temporarylocation= new GPSObject();
        temporarylocation.setTemp_CoordinationX(res.getDouble(res.getColumnIndex(COLUMN_COORDINATION_X)));
        temporarylocation.setTemp_CoordinationY(res.getDouble(res.getColumnIndex(COLUMN_COORDINATION_Y)));
        temporarylocation.setTemp_ACC(res.getDouble(res.getColumnIndex(COLUMN_ACC)));
        temporarylocation.setTimestamp(res.getLong(res.getColumnIndex(COLUMN_TIMESTAMP_GPS)));

        tempGPS.add(temporarylocation);
        res.moveToNext();
    }
    res.close();

    return tempGPS;
}
```

Figure 58 DBHelper - Location Stored Procedures

4.1.10 Update Service

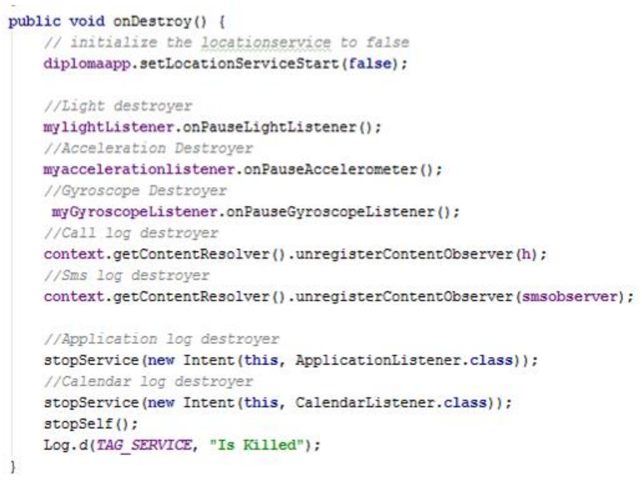
This module is responsible for triggering application listeners. User can start or stop the service in order to trigger the registration or deregistration of each listener. On the creation of the service we initialize all the proper parameters and register the listeners.



```
public class UpdateService extends Service {
    private static final String TAG_SERVICE = "MyService";
    AccelerometerListener myaccelerationlistener;
    LightListener mylightListener;
    GyroscopeListener myGyroscopeListener;
    GPSListener MyfirstLocation;
    Context context;
    CallListener h;
    SMSListener smsobserver;
    Uri uri = Uri.parse("content://sms");
    TestDiplomaApp diplomaapp;

    @Override
    public void onCreate() {
        context = this;
        diplomaapp=((TestDiplomaApp)context.getApplicationContext());
        //sms initialization
        smsobserver=new SMSListener(new Handler(), context);
        //Call initialization
        h=new CallListener(new Handler(), context);
        //Intent for location GPS
        MyfirstLocation = new GPSListener(context);

        if (MyfirstLocation.getLocation()==null) {
            Intent intent = new Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
            intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            startActivity(intent);
        }
        //Acceleration Service
        myaccelerationlistener = new AccelerometerListener(context);
        //Light Service
        mylightListener= new LightListener(context);
        //Gyroscope Service
        myGyroscopeListener = new GyroscopeListener(context);
        ///CallLogs Service
        context.getContentResolver().registerContentObserver(CallLog.Calls.CONTENT_URI, true, h);
        ///SMS Service
        context.getContentResolver().registerContentObserver(uri,true,smsobserver);
```

Figure 59 Update Service – Initialization & Registration of Listeners

When the service is stopped, automatically, all listeners are deregistered as the following figure represents.



```
public void onDestroy() {
    // initialize the locationservice to false
    diplomaapp.setLocationServiceStart(false);

    //Light destroyer
    mylightListener.onPauseLightListener();
    //Acceleration Destroyer
    myaccelerationlistener.onPauseAccelerometer();
    //Gyroscope Destroyer
     myGyroscopeListener.onPauseGyroscopeListener();
    //Call log destroyer
    context.getContentResolver().unregisterContentObserver(h);
    //Sms log destroyer
    context.getContentResolver().unregisterContentObserver(smsobserver);

    //Application log destroyer
    stopService(new Intent(this, ApplicationListener.class));
    //Calendar log destroyer
    stopService(new Intent(this, CalendarListener.class));
    stopSelf();
    Log.d(TAG_SERVICE, "Is Killed");
}
```

Figure 60 Update Service – Deregistration of Listeners

4.2 Context-awareness and device data

The core of the prototype application lies within the context manager module which is responsible for controlling the application’s rules as well as for carrying out all the appropriate actions that have already been programmed to provide context-awareness in user's device.

4.2.1 Context Rules

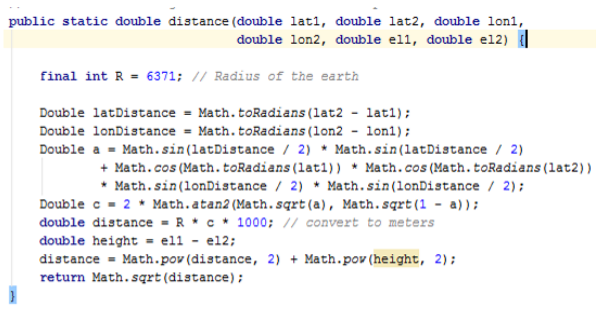
Rules are based upon the daily use of the device and are intended to help the user. Considering the continuous hustle that a regular user undergoes, we seek to release him from these routine movements. We defined ten simple rules:

1) If user is at home, then change the ringtone volume to mute and enable vibration: Detecting the smartphone’s location and the time of the departure we enable this rule in order to decrease the volume of the device and also enable the vibration.
2) If user is at home and time is over 23:59, then set device on Sleep Mode: Detecting the smartphone’s location and the time of the departure we enable this rule in order to mute the device and also disable the vibration.
3) If user is at office, then mute and enable vibration: Detecting the smartphone’s location, the time of the departure we enable this rule in order to mute the device and also enable the vibration.
4) If user is at meeting then set device on *Meeting Mode*: Detecting the smartphone’s location and the calendar events we enable this rule in order to mute the device and also enable the vibration. At the same time we disable the communication via calls.
5) If user leaves the office, then provide him a tool to organize his/hers commuting. Detecting the smartphone’s location and the exit time we enable this rule in order to create a connection with an external Google API to consume the current Public Transport directions based on user’s location. In case Public Transport is not available user can choose between other transport modes and/or another origin destination apart from her/his saved locations.
6) If user leaves the office, then notify him current events in city. Detecting the smartphone’s location and the exit time we enable this rule in order to create a connection with an external API to consume the events based on user’s location.

7) If a caller belongs to top three callers and Rule 4 is applied then notify caller via sms that user is at meeting. Detecting the call log file daily, we enable this rule in order to send a sms message inform caller that user is at meeting.
8) Every week display top three applications on homepage. Detecting the application log file weekly, we enable this rule in order to set on homepage the three most used applications
9) Every evening after work display the most used applications and contact name (for that datetime). Detecting the application and call log file daily, we enable this rule in order to set on homepage the most used application and contact profile.
10) If the light sensor is near zero and acceleration/speed different from zero then set the ringtone volume to 100% and enable vibration. Detecting the motion of the device the proximity sensor, the state of the device and the value of the light sensor periodically, we enable this rule in order to set the volume of ringtone to max value and set the vibration on.

4.2.2 ContextManager

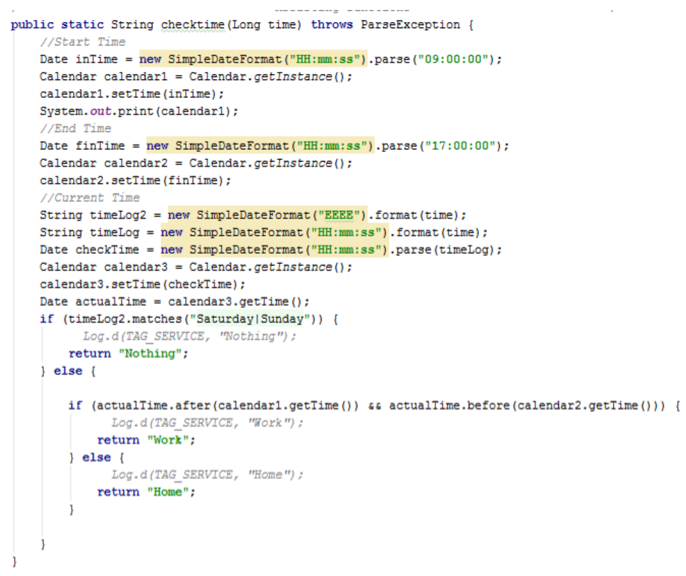
ContextManager is the module, which is responsible for the actions that trigger the execution of the aforementioned rules. It is composed of specific functions that oversees the collection of data and manages all the necessary checking procedures in order to provide context-awareness to mobile owner. The main function that helps to implement several rules is the *Geofencing* module. Here we generate a set of centroids which are selected based on a particular distance, is set at 300m. A comparison is applied on the difference of the initial position and the next position, if the difference is less than the certain distance then a new provisional center is defined by the mean value of the previous two items.



```java
public static double distance(double lat1, double lat2, double lon1,
                              double lon2, double el1, double el2) {

    final int R = 6371; // Radius of the earth

    Double latDistance = Math.toRadians(lat2 - lat1);
    Double lonDistance = Math.toRadians(lon2 - lon1);
    Double a = Math.sin(latDistance / 2) * Math.sin(latDistance / 2)
            + Math.cos(Math.toRadians(lat1)) * Math.cos(Math.toRadians(lat2))
            * Math.sin(lonDistance / 2) * Math.sin(lonDistance / 2);
    Double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    double distance = R * c * 1000; // convert to meters
    double height = el1 - el2;
    distance = Math.pow(distance, 2) + Math.pow(height, 2);
    return Math.sqrt(distance);
}
```

Figure 61 ContextManager – Distance Function

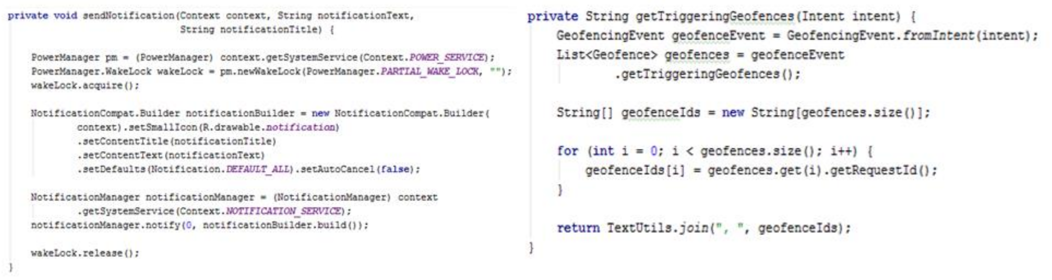
Thereby a number of centroids is calculated which is subsequently filtered in through the period and held two groups, the group that indicates the location "Home" and the second group that indicates the location "Work". For time filtering is used the *checktime* function which checks whether the timestamp of the stigma is between a specific time window.



```java
public static String checktime(Long time) throws ParseException {
    //Start Time
    Date inTime = new SimpleDateFormat("HH:mm:ss").parse("09:00:00");
    Calendar calendar1 = Calendar.getInstance();
    calendar1.setTime(inTime);
    System.out.print(calendar1);
    //End Time
    Date finTime = new SimpleDateFormat("HH:mm:ss").parse("17:00:00");
    Calendar calendar2 = Calendar.getInstance();
    calendar2.setTime(finTime);
    //Current Time
    String timeLog2 = new SimpleDateFormat("EEEE").format(time);
    String timeLog = new SimpleDateFormat("HH:mm:ss").format(time);
    Date checkTime = new SimpleDateFormat("HH:mm:ss").parse(timeLog);
    Calendar calendar3 = Calendar.getInstance();
    calendar3.setTime(checkTime);
    Date actualTime = calendar3.getTime();
    if (timeLog2.matches("Saturday|Sunday")) {
          Log.d(TAG_SERVICE, "Nothing");
        return "Nothing";
    } else {

        if (actualTime.after(calendar1.getTime()) && actualTime.before(calendar2.getTime())) {
              Log.d(TAG_SERVICE, "Work");
            return "Work";
        } else {
              Log.d(TAG_SERVICE, "Home");
            return "Home";
        }

    }
}
```

Figure 62 ContextManager - Checktime function

With the aim of the abovementioned functions it is calculated the user's location which triggers, depending on entering or exiting the relevant geographical areas, the actions that corresponds to specific rules. Simultaneously with these actions a notification message is prompt on device screen in order to notify the user for current changes. The above message is constructed by *sendNotification* and *getTriggeringGeofences* functions (Figure 63 ContextManager - Notification functions).

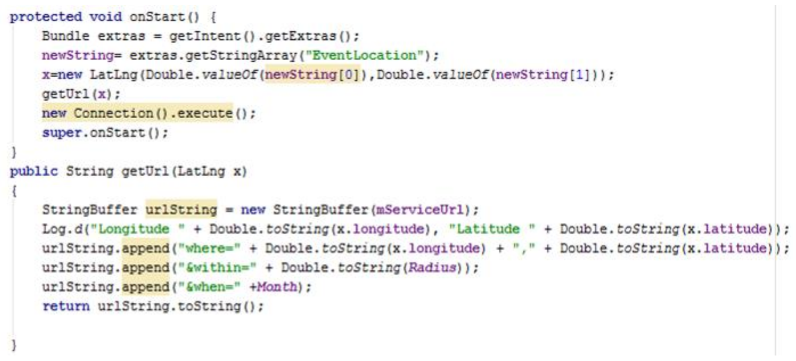


```java
private void sendNotification(Context context, String notificationText,
                              String notificationTitle) {

    PowerManager pm = (PowerManager) context.getSystemService(Context.POWER_SERVICE);
    PowerManager.WakeLock wakeLock = pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK, "");
    wakeLock.acquire();

    NotificationCompat.Builder notificationBuilder = new NotificationCompat.Builder(
            context).setSmallIcon(R.drawable.notification)
            .setContentTitle(notificationTitle)
            .setContentText(notificationText)
            .setDefaults(Notification.DEFAULT_ALL).setAutoCancel(false);

    NotificationManager notificationManager = (NotificationManager) context
            .getSystemService(Context.NOTIFICATION_SERVICE);
    notificationManager.notify(0, notificationBuilder.build());

    wakeLock.release();
}
```

```java
private String getTriggeringGeofences(Intent intent) {
    GeofencingEvent geofenceEvent = GeofencingEvent.fromIntent(intent);
    List<Geofence> geofences = geofenceEvent
            .getTriggeringGeofences();

    String[] geofenceIds = new String[geofences.size()];

    for (int i = 0; i < geofences.size(); i++) {
        geofenceIds[i] = geofences.get(i).getRequestId();
    }

    return TextUtils.join(", ", geofenceIds);
}
```

Figure 63 ContextManager - Notification functions

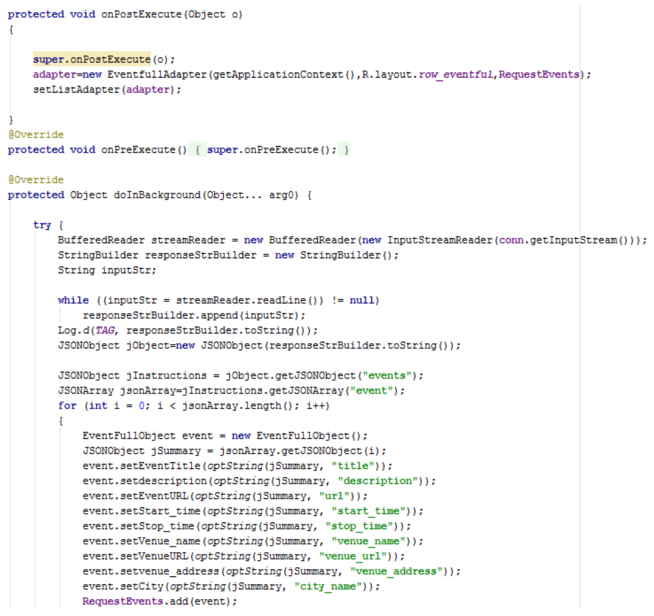
Another important module of *ContextManager* is the *Event* module. This module provides a dynamically search of current events based on user location and current date. It connects with an external API, namely Eventful API which is a worldwide known website[11] for upcoming events. *Event* module is also responsible for consuming the web service response which is implemented as an asynchronous task. Data conforms to JSON format and is converted to a readable format by the *EventAdapter*. The discrete steps of this process are first we build the URL (Figure 64 ContextManager – Building the HTTP request) for the web service request,



```java
protected void onStart() {
    Bundle extras = getIntent().getExtras();
    newString= extras.getStringArray("EventLocation");
    x=new LatLng(Double.valueOf(newString[0]),Double.valueOf(newString[1]));
    getUrl(x);
    new Connection().execute();
    super.onStart();
}
public String getUrl(LatLng x)
{
    StringBuffer urlString = new StringBuffer(mServiceUrl);
    Log.d("Longitude " + Double.toString(x.longitude), "Latitude " + Double.toString(x.latitude));
    urlString.append("where=" + Double.toString(x.longitude) + "," + Double.toString(x.latitude));
    urlString.append("&within=" + Double.toString(Radius));
    urlString.append("&when=" +Month);
    return urlString.toString();

}
```

Figure 64 ContextManager – Building the HTTP request

second we initialize a new HTTP connection, in our third step we consume the JSON format data and in our last step we display them as list view to the user.



```java
protected void onPostExecute(Object o)
{

    super.onPostExecute(o);
    adapter=new EventfullAdapter(getApplicationContext(),R.layout.row_eventful,RequestEvents);
    setListAdapter(adapter);

}
@Override
protected void onPreExecute() { super.onPreExecute(); }

@Override
protected Object doInBackground(Object... arg0) {

    try {
        BufferedReader streamReader = new BufferedReader(new InputStreamReader(conn.getInputStream()));
        StringBuilder responseStrBuilder = new StringBuilder();
        String inputStr;

        while ((inputStr = streamReader.readLine()) != null)
            responseStrBuilder.append(inputStr);
        Log.d(TAG, responseStrBuilder.toString());
        JSONObject jObject=new JSONObject(responseStrBuilder.toString());

        JSONObject jInstructions = jObject.getJSONObject("events");
        JSONArray jsonArray=jInstructions.getJSONArray("event");
        for (int i = 0; i < jsonArray.length(); i++)
        {
            EventFullObject event = new EventFullObject();
            JSONObject jSummary = jsonArray.getJSONObject(i);
            event.setEventTitle(optString(jSummary, "title"));
            event.setdescription(optString(jSummary, "description"));
            event.setEventURL(optString(jSummary, "url"));
            event.setStart_time(optString(jSummary, "start_time"));
            event.setStop_time(optString(jSummary, "stop_time"));
            event.setVenue_name(optString(jSummary, "venue_name"));
            event.setVenueURL(optString(jSummary, "venue_url"));
            event.setvenue_address(optString(jSummary, "venue_address"));
            event.setCity(optString(jSummary, "city_name"));
            RequestEvents.add(event);
```

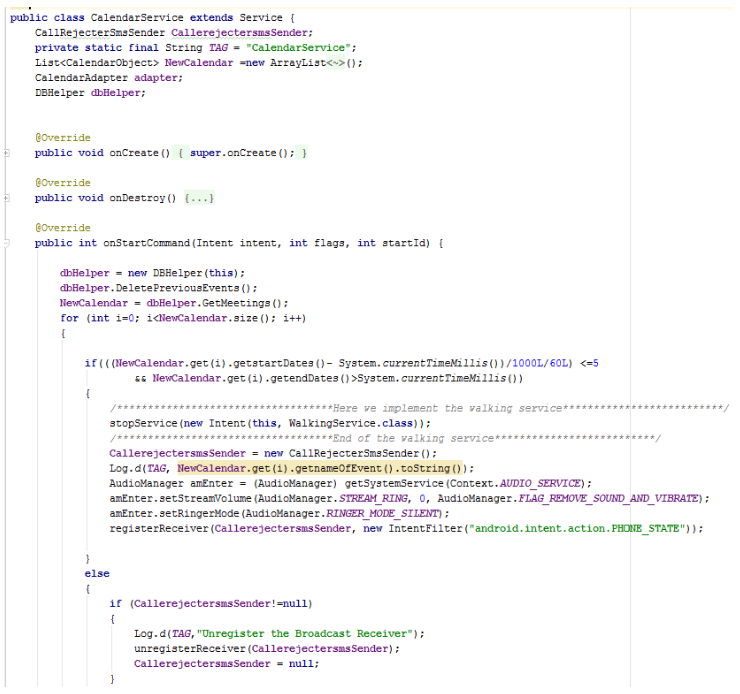
Figure 65 ContextManager – Web Service Consumption

Similar implementation with the *Event* module is the *PublicTransport* module. In this case we exploit the results of the *Geofencing* module and in accordance with the current time we provide to the user information regarding the public transport which operates

[11] www.eventful.com

on the defined locations. The user has the option to choose between the modes of transport (driving, walking, public transport) and between three pairs of location (Work-Home, Home-Work, Other). This module implements the external API of Google[12] and follows again specific steps. First we define the transport mode, second origin and destination, then we initialize a HTTP connection and then we call asynchronously *DirectionJSONParser* which parses the JSON information and provides the appropriate data in order to construct and visualize them on a map.

ContextManager also exploits the notion of upcoming calendar events via the *CalendarEvent* module. This module gathers all the relevant information which is stored in the database and examines the content of the event, the start / end time and compares with the current location and/or date time in order to trigger all the necessary predefined actions based on rules (Rule 7). In our case an action is defined as muting the device and switching off the vibration along with call forwarding except from the cases that a caller is one of user's top three callers and then a SMS message is notifying this caller that user is at a meeting. *ContentManager* initialize the Calendar Service module which notifies *CallRejecterSmsSender* function when a meeting is taking place.

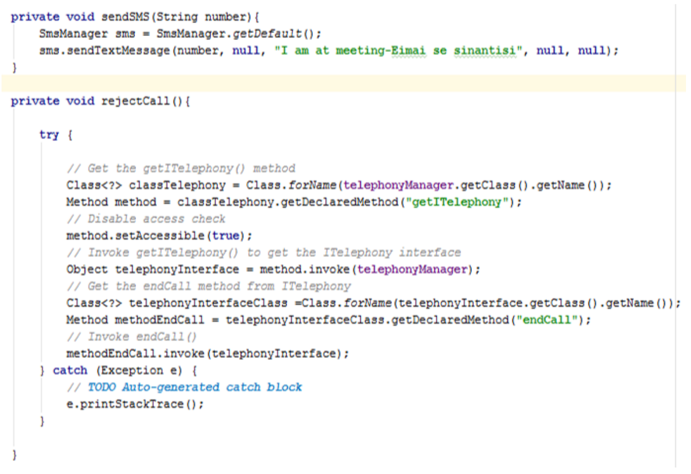


```
public class CalendarService extends Service {
    CallRejecterSmsSender CallerejectersmsSender;
    private static final String TAG = "CalendarService";
    List<CalendarObject> NewCalendar =new ArrayList<~>();
    CalendarAdapter adapter;
    DBHelper dbHelper;


    @Override
    public void onCreate() { super.onCreate(); }

    @Override
    public void onDestroy() {...}

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {

        dbHelper = new DBHelper(this);
        dbHelper.DeletePreviousEvents();
        NewCalendar = dbHelper.GetMeetings();
        for (int i=0; i<NewCalendar.size(); i++)
        {

            if(((NewCalendar.get(i).getstartDates()- System.currentTimeMillis())/1000L/60L) <=5
                    && NewCalendar.get(i).getendDates()>System.currentTimeMillis())
            {
                /**********************************Here we implement the walking service*************************/
                stopService(new Intent(this, WalkingService.class));
                /**********************************End of the walking service*************************/
                CallerejectersmsSender = new CallRejecterSmsSender();
                Log.d(TAG, NewCalendar.get(i).getnameOfEvent().toString());
                AudioManager amEnter = (AudioManager) getSystemService(Context.AUDIO_SERVICE);
                amEnter.setStreamVolume(AudioManager.STREAM_RING, 0, AudioManager.FLAG_REMOVE_SOUND_AND_VIBRATE);
                amEnter.setRingerMode(AudioManager.RINGER_MODE_SILENT);
                registerReceiver(CallerejectersmsSender, new IntentFilter("android.intent.action.PHONE_STATE"));

            }
            else
            {
                if (CallerejectersmsSender!=null)
                {
                    Log.d(TAG,"Unregister the Broadcast Receiver");
                    unregisterReceiver(CallerejectersmsSender);
                    CallerejectersmsSender = null;
                }
```

Figure 66 Context Manager – CalendarService

[12] https://developers.google.com/maps/documentation/directions/intro

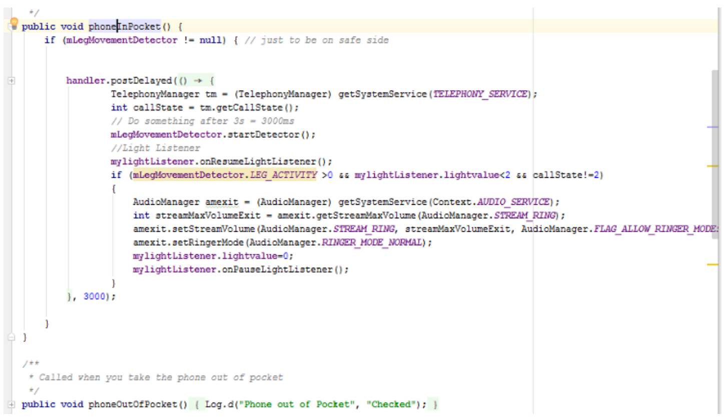
CallRejecterSmsSender handles the incoming call and in case this caller is identified as one of the top three callers then it notifies the caller with a SMS message.



```java
private void sendSMS(String number){
    SmsManager sms = SmsManager.getDefault();
    sms.sendTextMessage(number, null, "I am at meeting-Eimai se sinantisi", null, null);
}

private void rejectCall(){

    try {

        // Get the getITelephony() method
        Class<?> classTelephony = Class.forName(telephonyManager.getClass().getName());
        Method method = classTelephony.getDeclaredMethod("getITelephony");
        // Disable access check
        method.setAccessible(true);
        // Invoke getITelephony() to get the ITelephony interface
        Object telephonyInterface = method.invoke(telephonyManager);
        // Get the endCall method from ITelephony
        Class<?> telephonyInterfaceClass =Class.forName(telephonyInterface.getClass().getName());
        Method methodEndCall = telephonyInterfaceClass.getDeclaredMethod("endCall");
        // Invoke endCall()
        methodEndCall.invoke(telephonyInterface);
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

}
```

Figure 67 CallRejecterSmsSender - Reject call and send a SMS

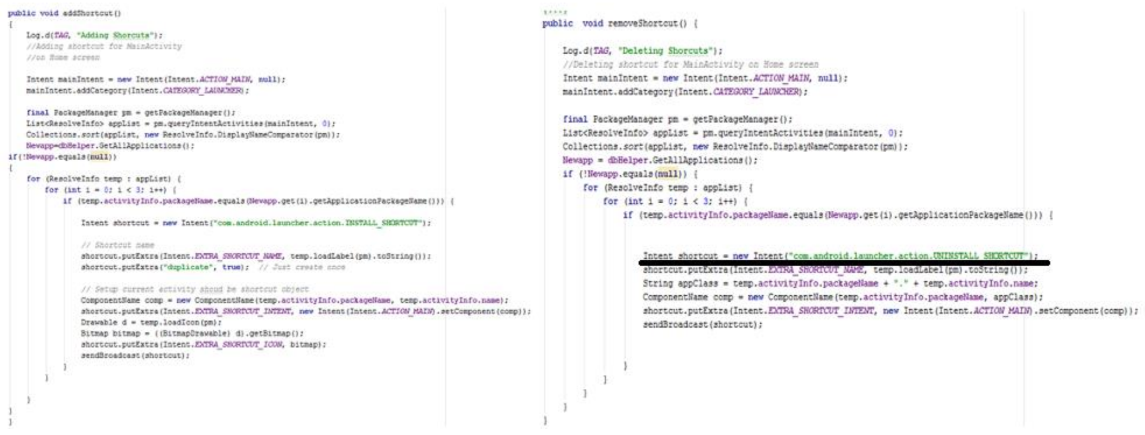
Furthermore ContextManager uses the data of acceleration sensor, data of light sensor and proximity sensor in order to find the movement of the holder and the position of the appliance via the *WalkingService* module. In case the user is moving and the device is fitted to some inner housing (pocket, bag etc) then this module triggers Rule 10. This module is registered when user exits a specific area and unregistered when user enters a specific area, in our case work and home location. *WalkingService* initialize the *PocketDetector*, a modification of *Artem Tartakynov* [52] code which checks whether a distance threshold is exceeded using proximity sensor and notifies *LegMovementDetector* which in turn counts the steps of the user. This function uses a common Kalman filter for smoothing the signal from accelerator. Finally *WalkingService* checks whether a leg movement is detected, when the value of light sensor is below 2 lux and the state of phone is different than off-hook (the condition that exists when a telephone is in use) and triggers changes to phone’s volume and vibration and also displaying a notification regarding the current changes.



```java
 */
public void phoneInPocket() {
    if (mLegMovementDetector != null) { // just to be on safe side


        handler.postDelayed(() -> {
                TelephonyManager tm = (TelephonyManager) getSystemService(TELEPHONY_SERVICE);
                int callState = tm.getCallState();
                // Do something after 3s = 3000ms
                mLegMovementDetector.startDetector();
                //Light Listener
                mylightListener.onResumeLightListener();
                if (mLegMovementDetector.LEG_ACTIVITY >0 && mylightListener.lightvalue<2 && callState!=2)
                {
                    AudioManager amexit = (AudioManager) getSystemService(Context.AUDIO_SERVICE);
                    int streamMaxVolumeExit = amexit.getStreamMaxVolume(AudioManager.STREAM_RING);
                    amexit.setStreamVolume(AudioManager.STREAM_RING, streamMaxVolumeExit, AudioManager.FLAG_ALLOW_RINGER_MODE:
                    amexit.setRingerMode(AudioManager.RINGER_MODE_NORMAL);
                    mylightListener.lightvalue=0;
                    mylightListener.onPauseLightListener();
                }
        }, 3000);

    }
}

/**
 * Called when you take the phone out of pocket
 */
public void phoneOutOfPocket() { Log.d("Phone out of Pocket", "Checked"); }
```

Figure 68 WalkingService - Phone in pocket

ContextManager hosts the *Shortcut* modules which are responsible for updating the home screen of user's device. It consists of two functions, one for inserting the shortcuts and the other one for deleting the shortcuts. In the first case we create a list of all the running applications and compare individually each application with the most used applications which are stored periodically to our database by the DBHelper. The second case in exactly the same as the first case with the only exception lies in the intent action. The following figure represents the aforementioned function highlighting the different parts.

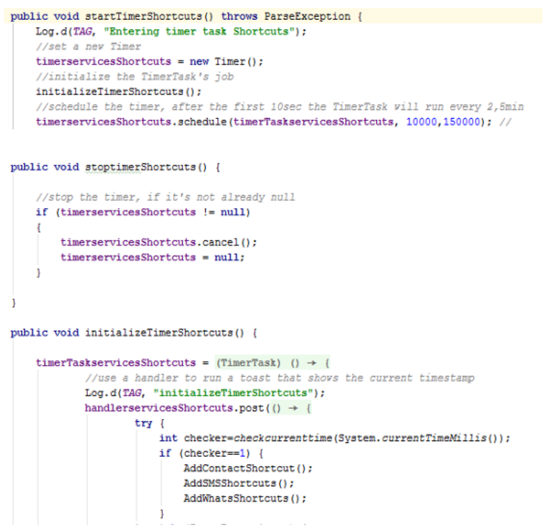


```java
public void addShortcut()
{
    Log.d(TAG, "Adding Shorcuts");
    //Adding shortcut for MainActivity
    //on Home screen

    Intent mainIntent = new Intent(Intent.ACTION_MAIN, null);
    mainIntent.addCategory(Intent.CATEGORY_LAUNCHER);

    final PackageManager pm = getPackageManager();
    List<ResolveInfo> appList = pm.queryIntentActivities(mainIntent, 0);
    Collections.sort(appList, new ResolveInfo.DisplayNameComparator(pm));
    Newapp=dbHelper.GetAllApplications();
if(!Newapp.equals(null))
{
    for (ResolveInfo temp : appList) {
        for (int i = 0; i < 3; i++) {
            if (temp.activityInfo.packageName.equals(Newapp.get(i).getApplicationPackageName())) {

                Intent shortcut = new Intent("com.android.launcher.action.INSTALL_SHORTCUT");

                // Shortcut name
                shortcut.putExtra(Intent.EXTRA_SHORTCUT_NAME, temp.loadLabel(pm).toString());
                shortcut.putExtra("duplicate", true);  // Just create once

                // Setup current activity shoud be shortcut object
                ComponentName comp = new ComponentName(temp.activityInfo.packageName, temp.activityInfo.name);
                shortcut.putExtra(Intent.EXTRA_SHORTCUT_INTENT, new Intent(Intent.ACTION_MAIN).setComponent(comp));
                Drawable d = temp.loadIcon(pm);
                Bitmap bitmap = ((BitmapDrawable) d).getBitmap();
                shortcut.putExtra(Intent.EXTRA_SHORTCUT_ICON, bitmap);
                sendBroadcast(shortcut);
            }
        }

    }
}
}
```

```java
*****/
public  void removeShortcut() {

    Log.d(TAG, "Deleting Shorcuts");
    //Deleting shortcut for MainActivity on Home screen
    Intent mainIntent = new Intent(Intent.ACTION_MAIN, null);
    mainIntent.addCategory(Intent.CATEGORY_LAUNCHER);

    final PackageManager pm = getPackageManager();
    List<ResolveInfo> appList = pm.queryIntentActivities(mainIntent, 0);
    Collections.sort(appList, new ResolveInfo.DisplayNameComparator(pm));
    Newapp = dbHelper.GetAllApplications();
    if (!Newapp.equals(null)) {
        for (ResolveInfo temp : appList) {
            for (int i = 0; i < 3; i++) {
                if (temp.activityInfo.packageName.equals(Newapp.get(i).getApplicationPackageName())) {


                    Intent shortcut = new Intent("com.android.launcher.action.UNINSTALL_SHORTCUT");
                    shortcut.putExtra(Intent.EXTRA_SHORTCUT_NAME, temp.loadLabel(pm).toString());
                    String appClass = temp.activityInfo.packageName + "." + temp.activityInfo.name;
                    ComponentName comp = new ComponentName(temp.activityInfo.packageName, appClass);
                    shortcut.putExtra(Intent.EXTRA_SHORTCUT_INTENT, new Intent(Intent.ACTION_MAIN).setComponent(comp));
                    sendBroadcast(shortcut);


                }
            }
        }
    }
}
```

Figure 69 ContextManager - Shortcut Module

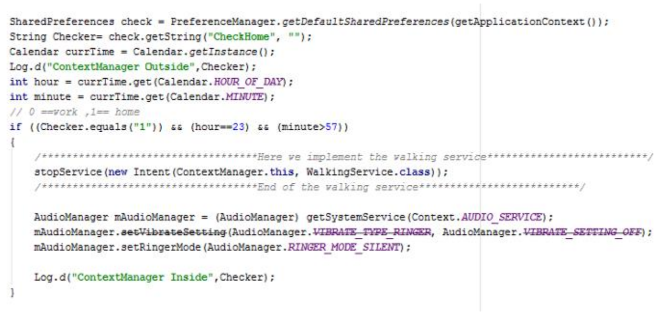
The same principals applied for the creation and deletion of phone contacts for calls and SMS shortcuts. In order to trigger the rules that related to shortcuts we implemented two specific time taskers. The first one is initialized to run every week and first delete the previous application shortcuts and second create new shortcuts based on the foreground duration of last week. The second one is initialized to run every day after work time and first remove previous contact shortcuts and then add the new ones based on the call duration and timestamp of the call. The added shortcuts (only if these contacts have the appropriate application accounts, namely if user has Hangouts or WhatsApp) can make a direct phone call, send a sms from Hangouts and also send a SMS from WhatsApp application. It is worth mentioning that in order to retrieve the contact’s image, the shortcut functions call the *RetrieveContactPhoto* and *Overlay* which retrieve from the device the image based on the contact number and overlay the retrieved image with application’s icon respectively.



```java
public void startTimerShortcuts() throws ParseException {
    Log.d(TAG, "Entering timer task Shortcuts");
    //set a new Timer
    timerservicesShortcuts = new Timer();
    //initialize the TimerTask's job
    initializeTimerShortcuts();
    //schedule the timer, after the first 10sec the TimerTask will run every 2,5min
    timerservicesShortcuts.schedule(timerTaskservicesShortcuts, 10000,150000); //


public void stoptimerShortcuts() {

    //stop the timer, if it's not already null
    if (timerservicesShortcuts != null)
    {
        timerservicesShortcuts.cancel();
        timerservicesShortcuts = null;
    }

}

public void initializeTimerShortcuts() {

    timerTaskservicesShortcuts = (TimerTask) () → {
            //use a handler to run a toast that shows the current timestamp
            Log.d(TAG, "initializeTimerShortcuts");
            handlerservicesShortcuts.post(() → {
                    try {
                        int checker=checkcurrenttime(System.currentTimeMillis());
                        if (checker==1) {
                            AddContactShortcut();
                            AddSMSShortcuts();
                            AddWhatsShortcuts();
                        }
```

Figure 70 ContextManager -TimeTasker for Contact shortcuts

Last but not least, *ContextManager* initialize another time tasker (*startTimerServices*) in order to implement Rule 2. This specific time tasker checks whether user is at home and also if time is near 00:00, to be precise it checks if time is after 23:57, and if this condition is true then it switch off the vibration and also mute the device.

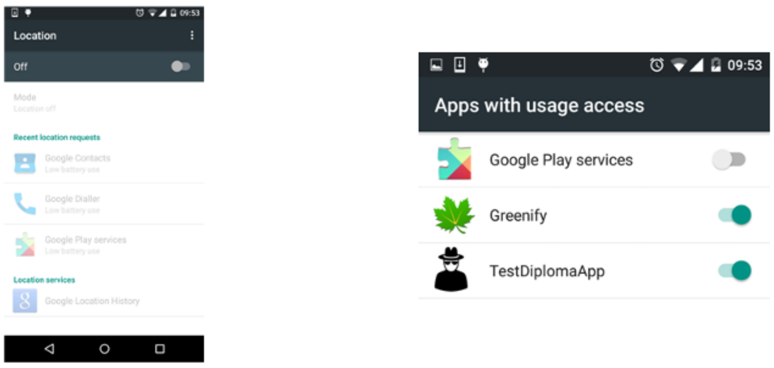


```java
SharedPreferences check = PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
String Checker= check.getString("CheckHome", "");
Calendar currTime = Calendar.getInstance();
Log.d("ContextManager Outside",Checker);
int hour = currTime.get(Calendar.HOUR_OF_DAY);
int minute = currTime.get(Calendar.MINUTE);
// 0 ==work ,1== home
if ((Checker.equals("1")) && (hour==23) && (minute>57))
{
    /**********************************Here we implement the walking service*************************/
    stopService(new Intent(ContextManager.this, WalkingService.class));
    /**********************************End of the walking service************************/

    AudioManager mAudioManager = (AudioManager) getSystemService(Context.AUDIO_SERVICE);
    mAudioManager.setVibrateSetting(AudioManager.VIBRATE_TYPE_RINGER, AudioManager.VIBRATE_SETTING_OFF);
    mAudioManager.setRingerMode(AudioManager.RINGER_MODE_SILENT);

    Log.d("ContextManager Inside",Checker);
}
```

Figure 71 ContextManager – startTimerServices (Rule 2)

5 Discussion

In this chapter we will discuss our results as described in previous sectors along with the functionalities of the prototype application. We will try to explain in full details the views (activities) of the application with the help of screenshots which have been taken during the test period in order to validate and assess the functioning of the Prototype application. Starting the application, the user needs to enable *Location* in order the application to receive GPS position. Additionally, in devices which are not rooted, another option appears that user should enable; *Apps with usage access* in order to allow the application to have control in a set of internal functionalities. In the following figures are displayed the aforementioned views.


09:53
Location
Off
Mode
Location off
Recent location requests
Google Contacts
Low battery use
Google Dialler
Low battery use
Google Play services
Low battery use
Location services
Google Location History

09:53
Apps with usage access
Google Play services
Greenify
TestDiplomaApp

Figure 72 Location and Usage Access Views

In the remainder of this chapter we will further analyze the components of the application that are divided in sensor's raw data views, the phone logger views and the Context Manager view.

5.1 Sensor's raw data Views

In these views are appearing all the raw data that the application fetches from device's sensors. For a clear view user can see exactly what have been saved in the database by pressing one of the four options depending on her preferences.

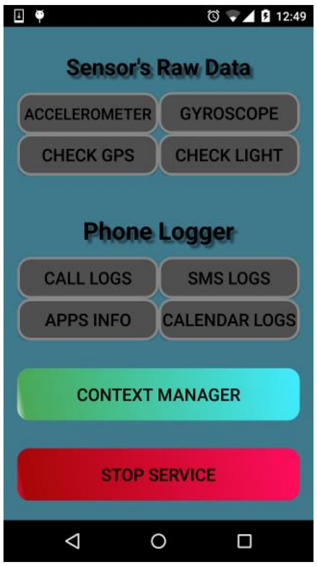
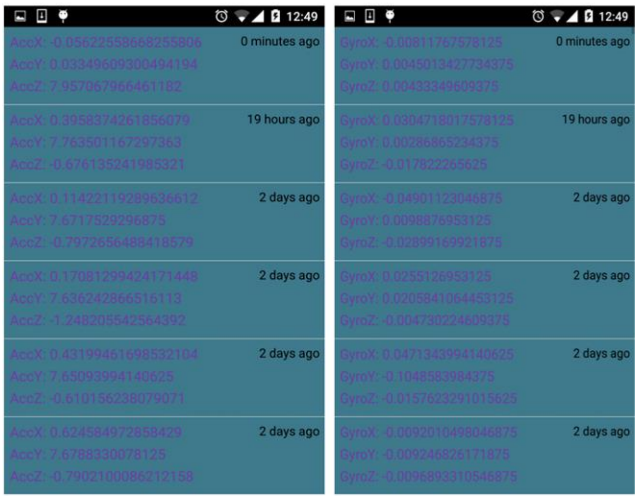


Figure 73 Prototype Homepage

By the time the homepage is displayed all the required listeners are registered in order to start the data gathering. In case user wants to unregister the listeners she can click to *Stop Service* button. Prototype will start gathering data the next time it will open.

User has the option to click on the four buttons which correspond to different sensors as indicated by their names. The following figure illustrates the raw data of accelerometer and gyroscope in 3-axis along with the timestamp of the record.

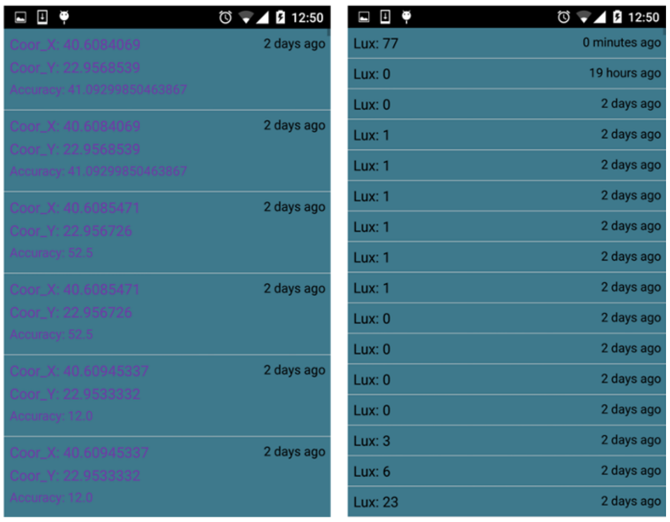


```
AccX: -0.05622558668255806        0 minutes ago
AccY: 0.03349609300494194
AccZ: 7.957067966461182
AccX: 0.3958374261856079          19 hours ago
AccY: 7.763501167297363
AccZ: -0.6761352419853821
AccX: 0.11422119289636612         2 days ago
AccY: 7.6717529296875
AccZ: -0.7972656488418579
AccX: 0.17081299424171448         2 days ago
AccY: 7.636242866516113
AccZ: -1.248205542564392
AccX: 0.43199461698532104         2 days ago
AccY: 7.65093994140625
AccZ: -0.6101562380790710
AccX: 0.624584972858429           2 days ago
AccY: 7.6788330078125
AccZ: -0.7902100086212158
```

```
GyroX: -0.0081176757578125        0 minutes ago
GyroY: 0.0045013427734375
GyroZ: 0.0043334960937 5
GyroX: 0.030471801757812 5        19 hours ago
GyroY: 0.0028686523437 5
GyroZ: -0.01782226562 5
GyroX: -0.04901123046875          2 days ago
GyroY: 0.0098876953125
GyroZ: -0.02899169921875
GyroX: 0.0255126953125            2 days ago
GyroY: 0.020584106445312 5
GyroZ: -0.0047302246093 75
GyroX: 0.04713439941406 25        2 days ago
GyroY: -0.104858398437 5
GyroZ: -0.01576232910156 25
GyroX: -0.009201049804687 5       2 days ago
GyroY: -0.00924682617187 5
GyroZ: -0.00968933105468 75
```

Figure 74 Raw data - Accelerometer and Gyroscope sensor

In the next figure is illustrated the GPS location (coordinates, position accuracy and the record's timestamp) and the also the light values in lux along with the record's timestamp.



12:50

Coor_X: 40.6084069 2 days ago
Coor_Y: 22.9568539
Accuracy: 41.09299850463867

Coor_X: 40.6084069 2 days ago
Coor_Y: 22.9568539
Accuracy: 41.09299850463867

Coor_X: 40.6085471 2 days ago
Coor_Y: 22.956726
Accuracy: 52.5

Coor_X: 40.6085471 2 days ago
Coor_Y: 22.956726
Accuracy: 52.5

Coor_X: 40.60945337 2 days ago
Coor_Y: 22.9533332
Accuracy: 12.0

Coor_X: 40.60945337 2 days ago
Coor_Y: 22.9533332
Accuracy: 12.0

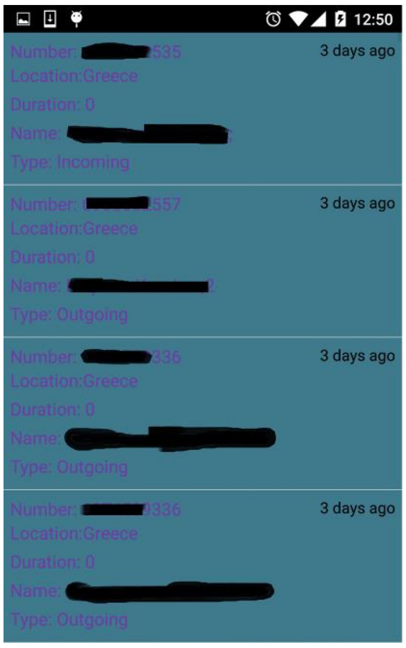
12:50

Lux: 77 0 minutes ago
Lux: 0 19 hours ago
Lux: 0 2 days ago
Lux: 1 2 days ago
Lux: 1 2 days ago
Lux: 1 2 days ago
Lux: 1 2 days ago
Lux: 1 2 days ago
Lux: 1 2 days ago
Lux: 0 2 days ago
Lux: 0 2 days ago
Lux: 0 2 days ago
Lux: 0 2 days ago
Lux: 3 2 days ago
Lux: 6 2 days ago
Lux: 23 2 days ago

Figure 75 Raw data - GPS and Light sensor

5.2 Phone Logger Views

In these views user can check what kind of information is saved regarding her calls, SMS messages, calendar events and also the installed applications on the device. In case user clicks on *Call Logs* button a new view will be displayed with the number of the caller, the location, the duration in minutes, the name of the caller and the type of call (incoming, outgoing).



12:50

Number: [redacted]535 3 days ago
Location:Greece
Duration: 0
Name: [redacted]
Type: Incoming

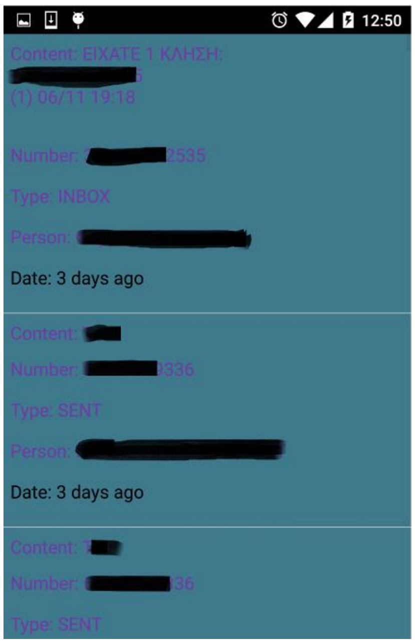
Number: [redacted]557 3 days ago
Location:Greece
Duration: 0
Name: [redacted]
Type: Outgoing

Number: [redacted]336 3 days ago
Location:Greece
Duration: 0
Name: [redacted]
Type: Outgoing

Number: [redacted]336 3 days ago
Location:Greece
Duration: 0
Name: [redacted]
Type: Outgoing

Figure 76 Phone Logger - Call Logs

In case user clicks on *Sms Logs* button, view will display the content of the message, the number of the sender, the type of message (inbox or sent), the sender's or receiver's name and the timestamp of the record.



12:50

Content: ΕΙΧΑΤΕ 1 ΚΛΗΣΗ:
[redacted]5
(1) 06/11 19:18

Number: [redacted]2535

Type: INBOX

Person: [redacted]

Date: 3 days ago

Content: [redacted]

Number: [redacted]336

Type: SENT

Person: [redacted]

Date: 3 days ago

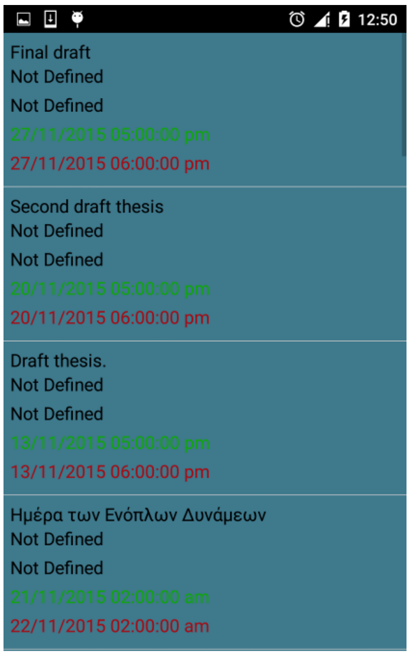
Content: [redacted]

Number: [redacted]36

Type: SENT

Figure 77 Phone Logger - Sms Logs

In case user clicks on *Calendar Logs* button, view will display the content, the description, the location, the starting date and the ending date of the event. If one of the previous values is not defined, then a specific message *Not Defined* is displayed in the corresponding position.



12:50

Final draft
Not Defined
Not Defined
27/11/2015 05:00:00 pm
27/11/2015 06:00:00 pm

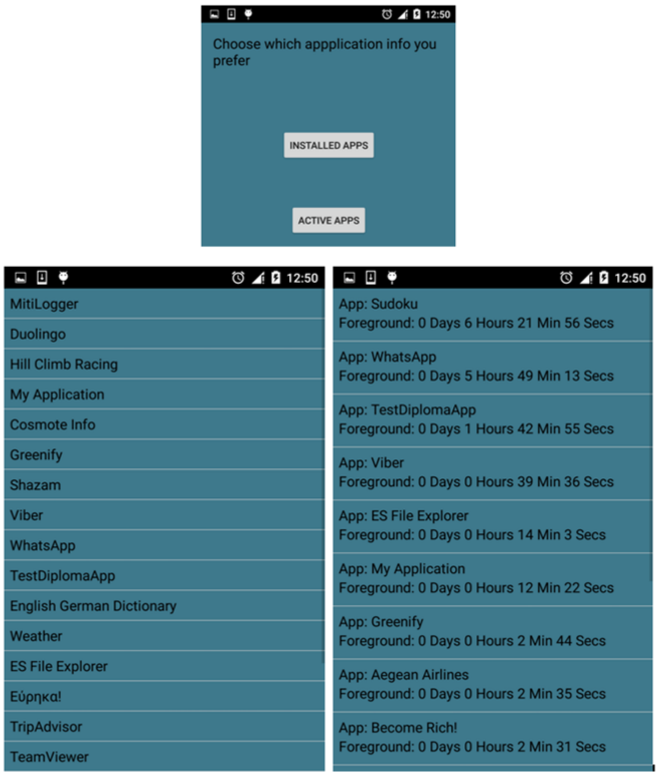
Second draft thesis
Not Defined
Not Defined
20/11/2015 05:00:00 pm
20/11/2015 06:00:00 pm

Draft thesis.
Not Defined
Not Defined
13/11/2015 05:00:00 pm
13/11/2015 06:00:00 pm

Ημέρα των Ενόπλων Δυνάμεων
Not Defined
Not Defined
21/11/2015 02:00:00 am
22/11/2015 02:00:00 am

Figure 78 Phone Logger - Calendar Logs

In case user clicks on *App Logs* button, two views can be displayed, one with the list of all the installed applications and the other one with the list of the applications that are in use along with the foreground duration of each one.


Choose which appplication info you prefer

MitiLogger
Duolingo
Hill Climb Racing
My Application
Cosmote Info
Greenify
Shazam
Viber
WhatsApp
TestDiplomaApp
English German Dictionary
Weather
ES File Explorer
Εύρηκα!
TripAdvisor
TeamViewer

App: Sudoku
Foreground: 0 Days 6 Hours 21 Min 56 Secs
App: WhatsApp
Foreground: 0 Days 5 Hours 49 Min 13 Secs
App: TestDiplomaApp
Foreground: 0 Days 1 Hours 42 Min 55 Secs
App: Viber
Foreground: 0 Days 0 Hours 39 Min 36 Secs
App: ES File Explorer
Foreground: 0 Days 0 Hours 14 Min 3 Secs
App: My Application
Foreground: 0 Days 0 Hours 12 Min 22 Secs
App: Greenify
Foreground: 0 Days 0 Hours 2 Min 44 Secs
App: Aegean Airlines
Foreground: 0 Days 0 Hours 2 Min 35 Secs
App: Become Rich!
Foreground: 0 Days 0 Hours 2 Min 31 Secs

Figure 79 Phone Logger - App Logs

5.3 Context Manager View

The initial view of the Context Manager is consisted of specific buttons which are responsible for specific functionalities. The following figure illustrates the abovementioned buttons which will be described later on this sub chapter.

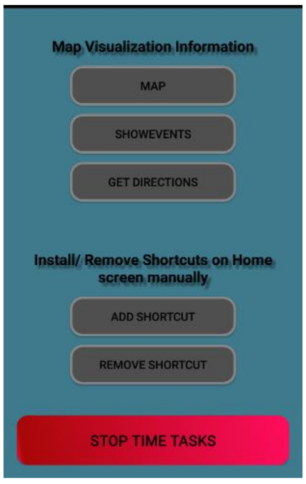
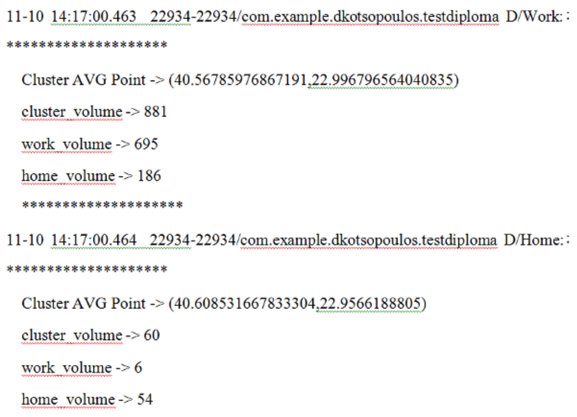

Map Visualization Information
MAP
SHOWEVENTS
GET DIRECTIONS
Install/ Remove Shortcuts on Home screen manually
ADD SHORTCUT
REMOVE SHORTCUT
STOP TIME TASKS

Figure 80 Context Manager View

As we have already mentioned Context Manager is responsible for the data management and all the required actions to be performed in the event when specific rules are valid. The moment user clicks *Context Manager* button all the appropriate listeners and time taskers are registered and initialized respectively. This moment is also the time when *Work* and *Home* locations are specified and geofencing process is activated. The user can see the results of this process by clicking the *Map* button. In the next figures are illustrated the background log (internal) and the visualization of *Work* and *Home* positions.



```
11-10 14:17:00.463   22934-22934/com.example.dkotsopoulos.testdiploma D/Work: :
********************
   Cluster AVG Point -> (40.56785976867191,22.996796564040835)
   cluster_volume -> 881
   work_volume -> 695
   home_volume -> 186
   ********************
11-10 14:17:00.464   22934-22934/com.example.dkotsopoulos.testdiploma D/Home: :
********************
   Cluster AVG Point -> (40.608531667833304,22.9566188805)
   cluster_volume -> 60
   work_volume -> 6
   home_volume -> 54
```

Figure 81 Context Manager – Background Geofencing Process

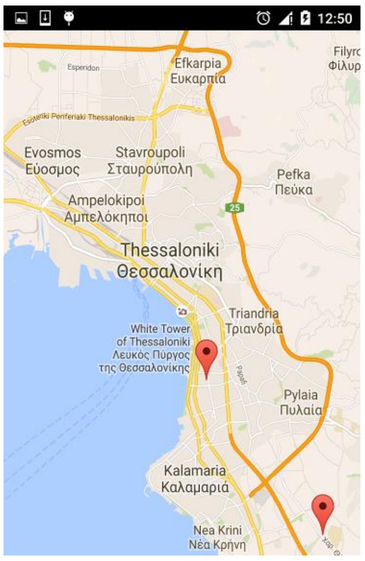


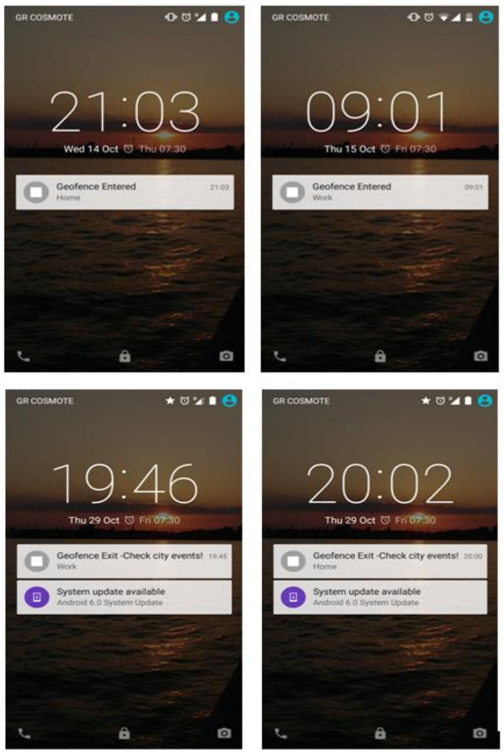
Figure 82 Context Manager – Visualization of Geofencing Process

When user enters or exits the geofencing area specific notifications are displayed along with the appropriate processes that were triggered based on the Rules, namely:

a) When a user enter the work position Rule number 3 is triggered;

b) When user exits the work position Rule number 5 is triggered;

c) When user enter the home position Rule number 1 is triggered.


GR COSMOTE
21:03
Wed 14 Oct Thu 07:30
Geofence Entered 21:03
Home

GR COSMOTE
09:01
Thu 15 Oct Fri 07:30
Geofence Entered 09:01
Work

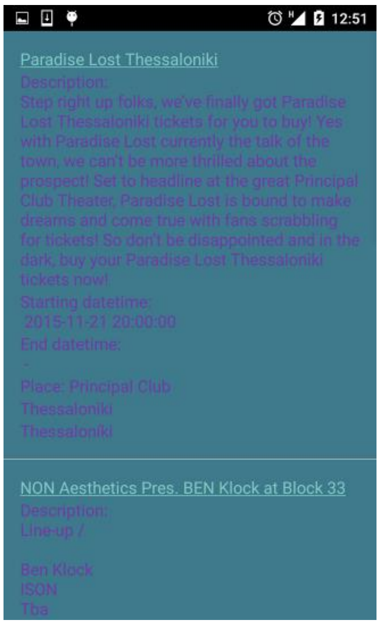
GR COSMOTE
19:46
Thu 29 Oct Fri 07:30
Geofence Exit -Check city events! 19:45
Work
System update available
Android 6.0 System Update

GR COSMOTE
20:02
Thu 29 Oct Fri 07:30
Geofence Exit -Check city events! 20:00
Home
System update available
Android 6.0 System Update

Figure 83 Context Manager – Notifications

It is worth mentioning that in case use clicks on notification which indicates to check city events then she/he will be redirected to the Event view which is discussed below.

In case user clicks on *ShowEvents* button or the appropriate notification, the application is automatically connected to the external Eventful API and fetches the appropriate event records in order to display them in a readable form. The available information regarding the event is the location of the event, the description with the starting and ending date of the event, the area where the event will take place. In case such information is not available then a (-) will be displayed.



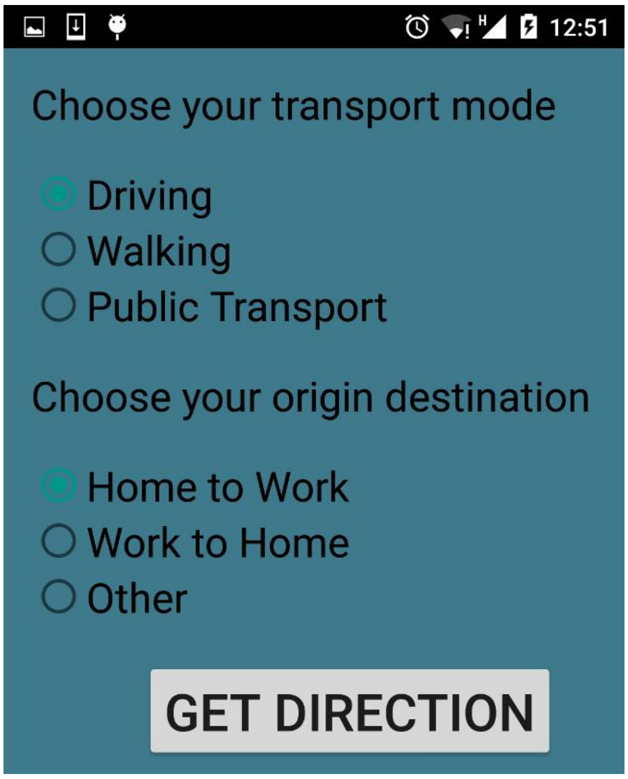
Paradise Lost Thessaloniki
Description:
Step right up folks, we've finally got Paradise Lost Thessaloniki tickets for you to buy! Yes with Paradise Lost currently the talk of the town, we can't be more thrilled about the prospect! Set to headline at the great Principal Club Theater, Paradise Lost is bound to make dreams and come true with fans scrabbling for tickets! So don't be disappointed and in the dark, buy your Paradise Lost Thessaloniki tickets now!
Starting datetime:
2015-11-21 20:00:00
End datetime:
-
Place: Principal Club
Thessaloniki
Thessaloniki

NON Aesthetics Pres. BEN Klock at Block 33
Description:
Line-up /

Ben Klock
ISON
Tba

Figure 84 Context Manager - Show Events

In case user exit her office or work and want to organize her commuting (Rule 6) then user can click on *Get Directions* button and be redirected to another view in order to choose the transport mode and the origin and destination. Except from the two specific routes (Home-Work, Work-Home) user can choose *Other* and select on the map another origin/destination and get directions and visualization for this route.



Choose your transport mode

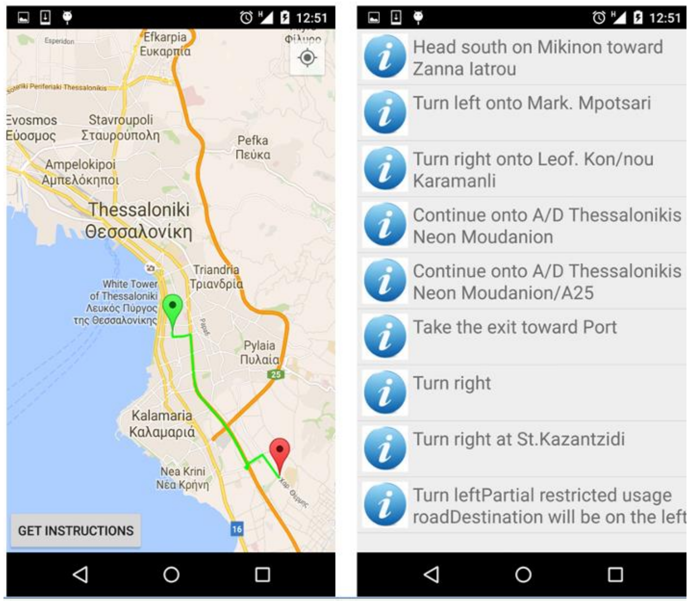
(●) Driving
( ) Walking
( ) Public Transport

Choose your origin destination

(●) Home to Work
( ) Work to Home
( ) Other

Figure 85 Context Manager - Directions

In any case users can see the results either as instructions or as a line on the map.



Head south on Mikinon toward Zanna Iatrou

Turn left onto Mark. Mpotsari

Turn right onto Leof. Kon/nou Karamanli

Continue onto A/D Thessalonikis Neon Moudanion

Continue onto A/D Thessalonikis Neon Moudanion/A25

Take the exit toward Port

Turn right

Turn right at St.Kazantzidi

Turn leftPartial restricted usage roadDestination will be on the left

Figure 86 Context Manager – Results

Furthermore, two more buttons are displayed regarding shortcuts. This functionality (Rule 8&9) is triggered by the time user clicks on *Context Manager* button. We have provided to user the possibility to also handle this process manually. In case user clicks on *Add Shortcut* button then all the appropriate shortcuts (Application, Phone, SMS and WhatsApp contacts) are installed in home screen. In case user clicks on *Delete Shortcut* then the reverse process is triggered and all the installed shortcuts are deleted. The following figure displays the results of the click actions.

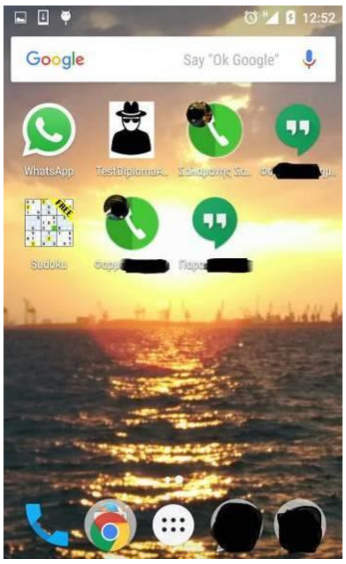
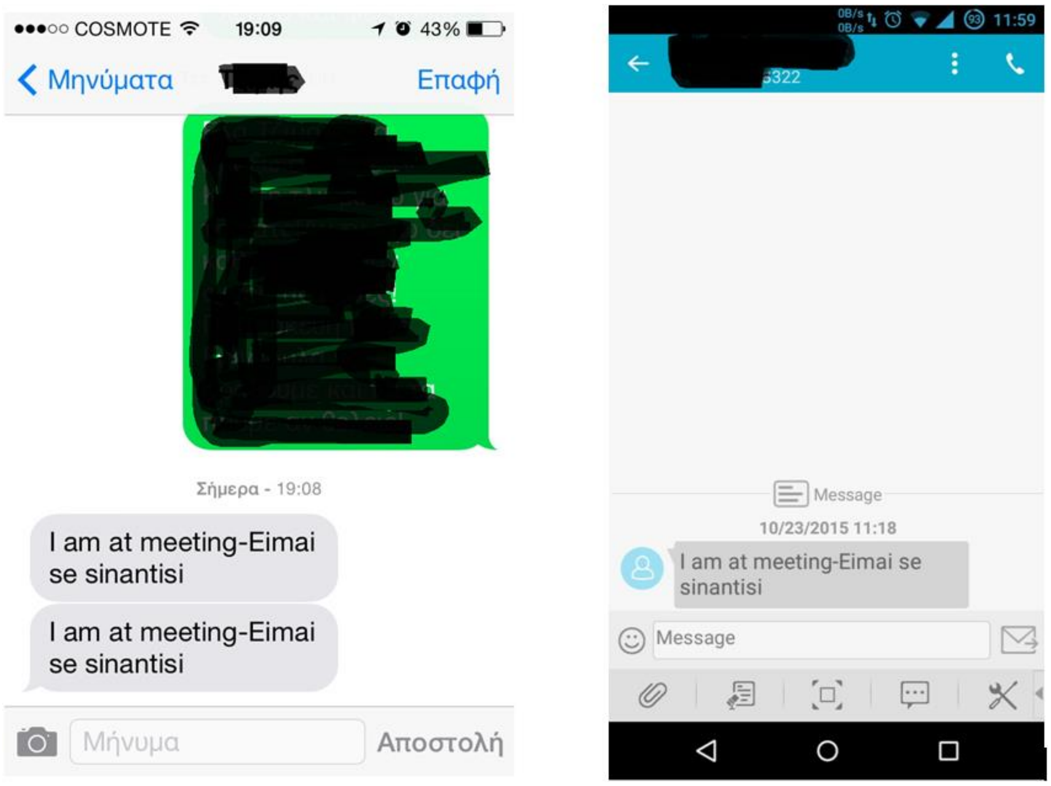


Figure 87 Context Manager - Add Shortcuts

The last button that is displayed on the Context Manager view is the *Stop Time Tasks* button. In case user clicks on this specific button then all the initialized time taskers are deactivated. The referred time taskers are three: the first is related to weekly shortcut installments; the second to daily shortcut installment and the last one is responsible for checking whether user is at home and time is after 23:57 in order to implement Rule 2.

Finally we should mention that two specific listeners that were registered during the initialization of the Context Manager view continue working incessantly and the first is notifying the user when Rule 4&7 are verified. In such cases all calls are rejected and only if a caller belongs to top three callers will be notified that user is unable to be reached. The following figure illustrates such SMS notifications in testing period.



Μηνύματα [redacted] Επαφή

[redacted]

Σήμερα - 19:08

I am at meeting-Eimai se sinantisi

I am at meeting-Eimai se sinantisi

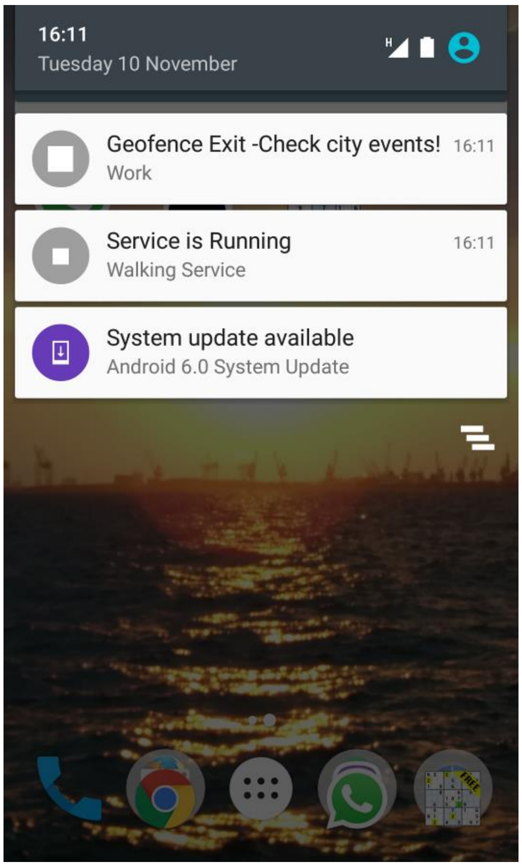
[redacted]5322

10/23/2015 11:18

I am at meeting-Eimai se sinantisi

Figure 88 Context Manager - Sms meeting Notifications

The second listener is responsible for enabling the Walking service and notifies the user when this service has started (Figure 89 Context Manager - Walking Service Notification). While this service is on, if user places her device into a pocket then it triggers Rule 10.


16:11
Tuesday 10 November

Geofence Exit -Check city events! 16:11
Work

Service is Running 16:11
Walking Service

System update available
Android 6.0 System Update

Figure 89 Context Manager - Walking Service Notification

As stated in Chapter 3, the conception and implementation of the application architecture is influenced by the Context Toolkit [31] and SOCAM [13] with a spirit to exploit as much as device sensors which were not available in previous efforts which were described in chapter 2. During our application development we focused on finding the most popular locations of the users (Work/Home) with a view to device management both indoor and outdoor of these locations in contrast with Active Badge [22] which focused on the call forwarding in specific locations inside the work perimeters. Personal Shopping Assistant [24] and Cyberguide [25] researchers tried to user personal preferences in order to assist them in specific occasions such as shopping or sightseeing whereas we concentrated on providing assistance on daily routine (daily events, public transport, customized home screen etc.). Furthermore we tried through the communication log and calendar log to inform callers about the desired specific situation (meeting mode) in contrast with HEP [26] and Office Assistant [27] where the application informed all contacts provided the proper communication means or offer information to one user for a specific occupied/unoccupied office respectively. None of the aforementioned applications focused on motion state aspect of context-awareness as we did, where the device based on the specific location (Outdoor and in pocket) and movement (walk) aims the user by maximizing the volume of the device when walking outdoor.

6 Conclusions and Future Work

6.1 Synopsis

In this dissertation we have attempted to present to reader the concepts of context and context-awareness and how a user of a mobile phone can benefit using applications with such technologies. We made a historical review of a variety of research efforts which were the solid foundations for the further development of this technology aspect. We saw that through specific design methods we can create architectures which can easily be extended and managed according to the needs of ordinary users and we continue by analyzing the system architecture and application development tools we used and we came into the detailed analysis of subcomponents and the background services. Additionally we defined the rules that Context Manager of the prototype application have to manage and the results of these actions. Finally we presented the created prototype application as a proof of concept trying to elaborate that it is possible to analyze the user's profile through daily performed operations in order to provide personalized services that assist the user. It is worth mentioning that during the dissertation three major IT companies, Google, Nokia and Microsoft[13] have presented as beta versions or stable releases application-launchers which more or less provide similar functionalities (Add/delete shortcuts etc) with our prototype. This indicates the trend that will be followed from IT companies based on more personalized services and customization with the aim of context-awareness. The research area concerned the context-awareness computing is and will continue to be fruitful for research and development of new functionalities or software components over the upcoming years combining contextual information from people, processes, data and things under the umbrella of Internet of Everything.

[13] Marshmallow Android, Nokia Z Launcher, Microsoft Arrow Launcher

6.2 Conclusions

During the application's development we used specific tools for programming and debugging in Android Environment such as Android Studio as well as smartphone Nexus 5 (Google/LG) for testing reasons, which was the basis for the development of the prototype application. We proved the concept that based on specific sensors and user's call and SMS log a satisfactory level of context-awareness can be achieved. For this reason we were oriented to this specific smartphone model emulator as well as the Android OS 5.1 version. We believe the choices that have been made regarding the embedded rules and also the processes that triggered the predefined rules provide a decent context-aware prototype which helps the daily routine of the user without creating additional problems than user already has. The prototype application is designed for minimum interaction between user and device due to the fact that most of the functionalities are background services, for example the algorithm for generating work and home locations is triggered every time the *Context Manager* button is clicked instead of asking user to define her preferences. Our intention was user to forget during the day that has a running application in the smartphone, but only perceive it at those moments that the application will inform her via notifications.

Through the realization of the application we resulted that the acceleration and light sensor are valuable source of information in contrast to gyroscope sensor which in our case the raw data have worthless information value, unless used in conjunction with data from the accelerometer sensor with a purpose to optimize the movement recognition of the device which at this stage is not necessary. Furthermore it came to our notice that it is not needed to record the content of the messages but sufficient to record the number of messages and their timestamp from/to user to/from a contact.

However, during the thesis we encountered many difficulties regarding prototype's compatibility and operability:

- The range of mobile devices that uses Android OS is vast as vast it is the diversification over the same operation system that varies from company to company;
- The development of the application was based on versions 5 and above resulting that many functions of the application could not operate correctly on lower versions;

- Most previous programming functionalities that an application developer had in newer versions presented inconsistency and they did not function as they functioned in versions such as 4.4, namely deletion of the shortcuts which in previous versions was working without errors where as in version 5.0 and above the Google launcher forbade us to delete shortcuts on home screen;
- Many mobile devices, in most cases older models, do not incorporate all the used sensors which the prototype application uses in order to activate / deactivate rules and essentially manage the data depending on the active processes.

Nevertheless in this diploma thesis, we came to the conclusion that not only it is possible to create a context-aware application based on simple design methodology using as essential information data from mobile sensors and user's communication log but the outcome of this dissertation, the prototype android application, corroborate it.

6.3 Future Work

As mentioned in Conclusions we encountered various issues regarding the compatibility of application in various smartphones, Android OS releases and versions of Android. It is completely subjective choice of the developers to decide which Android version will be the target group in order to focus and solve or find workarounds for the aforementioned issues. Furthermore, dependence of the application of the user command can be considered again as a subjective choice and it is up to the developer the degree of application independence from individual choices of users.

The steps need to be taken to release a stable beta version for the prototype application with a view to the evaluation from a sufficient number of users is the following:

- Stabilize the prototype application. We could complete a circle of given tests to make sure that the application works under different conditions without displaying error that could terminate its operation;
- Manage efficiently the SQL insertion/deletion/fetching queries. We could create a specific service that manages the input / output of data with pre / post and background tasks to achieve an optimal management of the device's resources;
- Decrease the battery consumption by handling more efficiently asynchronous task. We could monitor the procedures and mark the most hungry-tasks in order

to manage them accordingly, for example gathering GPS values on specific time or planning the execution of some queries when mobile is on idle mode etc.;

- Define the target group of the Android versions and create some workarounds for deprecated or problematic functionalities (Delete Shortcuts). We could monitor the mobile market and see the upcoming trends of mobile market and choose based on this research the largest share market to satisfy;
- Create a preference view in order to define specific settings (volume, vibration, duration for time taskers) to make it more customized for each user. We could create a specific activity/view in order to customize specific actions that triggered from the rules. For example one user may want to turn the vibration and volume off when at Work whereas the same time another user may desire to have the volume up.

Bibliography

[1] Gartner Group, Top 10 Strategic Technology Trends for 2013

[2] IDC, IDC Predictions 2015: Accelerating Innovation and Growth on the 3rd Platform

[3] M.Weiser, The Computer for the 21st Century 1991

[4] P. Guillemin and P. Friess, Internet of things strategic research roadmap, The Cluster of European Research Projects, Tech. Rep., September 2009

[5] C Perera, A Zaslavsky, P Christen, D Georgakopoulos, Context Aware Computing for The Internet of Things: A Survey - Communications Surveys & Tutorials, IEEE, 2014

[6] B. Schilit, N. Adams and R Want, Context-aware computing applications, 1994

[7] P.J. Brown, J.D. Bovey and Xian Chen, Context-aware applications: From the laboratory to the marketplace. IEEE Personal Communications 4(5): pp. 58-64.October 1997

[8] Anind K. Dey, Providing Architectural Support for Building Context-Aware Applications, 2000

[9] B. Schilit and Theimer, Disseminating Active Map Information to Mobile Hosts 1994

[10] J. Pascoe, Adding generic contextual capabilities to wearable computers, 1998

[11] G.D. Abowd, A.K. Dey, P.J. Brown, N. Davies, M. Smith and P. Steggles, Towards a better understanding of context and context-awareness, 1999

[12] H. Chen, T. Finin, A. Joshi, L. Kagal, F. Perich, and D. Chakraborty, Intelligent agents meet the semantic web in smart spaces, 2004

[13] T. Gu, H. K. Pung, and D. Q. Zhang, A service-oriented middleware for building context-aware services, 2004

[14] M. Smith, C. Welty, D. McGuinness, Web Ontology Language (OWL) Guide, 2003.

[15] L. Capra, W. Emmerich, and C. Mascolo, Carisma: context-aware reflective middleware system for mobile applications, 2003

[16] M. Roman, C. Hess, R. Cerqueira, A. Ranganat, R. H. Campbell and K. Nahrstedt, Gaia: A Middleware Infrastructure to Enable Active Spaces, 2002

[17] G. E. Krasner, S. T. Pope, A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System, 1988

[18] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste, Project aura: Toward distraction-free pervasive computing, 2002

[19] Fei Li, S. Sehic, S. Dustdar, COPAL: An adaptive approach to context provisioning, October 2010

[20] Perera, C., Zaslavsky, A., Christen, P., Georgakopoulos, D. Ca4iot: Context awareness for internet of things. In Proceedings of the 2012 IEEE International Conference on Green Computing and Communications, Besançon, France, 20–23 November 2012; IEEE Computer Society: Washington, DC, USA, 2012; pp. 775–782

[21] Huertas Celdran, A., Garcia Clemente, F.J.; Gil Perez, M.; Martinez Perez, G. SeCoMan: A Semantic-Aware Policy Framework for Developing Privacy-Preserving and Context-Aware Smart Applications. IEEE Syst. J. 2013

[22] R. Want, A. Hopper, V. Falcao and J. Gibbons, The active badge location system, 1992

[23] R. Want, B. N. Schilit, N. I. Adams, R. Gold, K. Petersen, D. Goldberg, J. R. Ellis and M. Weiser, An Overview of the ParcTab Ubiquitous Computing Experiment, 1995

[24] A. Asthana, M. Cravatts, and P. Krzyzanouski, An indoor wireless system for personalized shopping assistance, 1994

[25] Gregory D. Abowd, Christopher G. Atkeson, Jason Hong, Sue Long, Rob Kooper and Mike Pinkerton, Cyberguide: A Mobile Context-Aware Tour Guide, 1996

[26] B. Chihani, E. Bertin, F. Jeanne and N. Crespi HEP: context-aware communication system, 2011

[27] H. Yan, T. Selker, Context-Aware Office Assistant, In Proceedings of the 2000 International Conference on Intelligent User Interfaces, 2000

[28] M. M. Molla and S. I. Ahamed, A survey of middleware for sensor network and challenges, in Proceedings of the 2006 International Conference Workshops on Parallel Processing ,2006

[29] K. E. Kjaer, A survey of context-aware middleware, in Proceedings of the 25th conference on IASTED International Multi-Conference, 2007

[30] M. Baldauf, S. Dustdar, and F. Rosenberg, A survey on context aware systems, Int. J. Ad Hoc Ubiquitous Computing, 2007

[31] Anind K. Dey and Gregory D. Abowd, The Context Toolkit: Aiding the Development of Context-Aware Applications, 2000

[32] G. Chen and D. Kotz, A survey of context-aware mobile computing research, 2000

[33] Hevner, A.R., March, S.T., and Park, J. Design Research in Information Systems Research. MIS Quarterly, 28, 1 (2004), 75-105

[34] Takeda H. ,Veerkamp, P., Tomiyama, T., and Yoshikawam, H. (1990). Modeling Design Processes." AI Magazine 1990 Winter

[35] Fann, K. T. Peirce's Theory of Abduction. The Hague, The Netherlands: 1970.

[36] Android Developers Blog, Accessed August 20/2015, http://android-developers.blogspot.in

[37] Martin Gudgin, Marc Hadley, Jean-Jacques Moreau, Henrik Frystyk Nielsen, www.w3.org, Accessed August 2/2015, http://www.w3.org/TR/2001/WD-soap12-part1-20011217

[38] Android Developers, developer.android.com, Accessed August 13/2015, http://developer.android.com/reference/android/location/LocationManager.html

[39] Android Developers, developer.android.com, Accessed August 13/2015, http://developer.android.com/reference/android/location/LocationListener.html

[40 Android Developers, developer.android.com, Accessed August 22/2015, http://developer.android.com/reference/android/hardware/Sensor.html

[41] Android Developers, developer.android.com, Accessed August 22/2015, http://developer.android.com/reference/android/hardware/SensorEvent.html

[42] Android Developers, developer.android.com, Accessed August 23/2015, http://developer.android.com/reference/android/hardware/SensorEventListener.html

[43] Android Developers, developer.android.com, Accessed August 25/2015, http://developer.android.com/reference/android/provider/CallLog.html

[44] Android Developers, developer.android.com, Accessed August 20/2015, http://developer.android.com/reference/android/provider/CallLog.Calls.html

[45] Android Developers, developer.android.com, Accessed August 20/2015, http://developer.android.com/reference/android/database/ContentObserver.html

[46] Android Developers, developer.android.com, Accessed September 20/2015, https://developer.android.com/reference/android/provider/Telephony.Sms.html

[47] Android Developers, developer.android.com, Accessed October 20/2015, http://developer.android.com/reference/android/provider/ContactsContract.html

[48] Android Developers, developer.android.com, Accessed October 20/2015, http://developer.android.com/reference/android/content/pm/PackageManager.html

[49] Android Developers, developer.android.com, Accessed September 15/2015, https://developer.android.com/reference/android/app/usage/UsageStatsManager.html

[50] Android Developers, developer.android.com, Accessed October 20/2015, http://developer.android.com/guide/components/services.html

[51] Android Developers, developer.android.com, Accessed October 20/2015, http://developer.android.com/guide/topics/providers/calendar-provider.html

[52] Tartakynov Artem, github.com, Accessed October 10/2015

https://github.com/tartakynov/robowalk/blob/master/src/com/tartakynov/robotnoise/PocketDetector.java

Appendix

In this chapter we present the most significant part of the source code, namely Context Manager. In case readers seek the rest of the source code they can find it in the following link.

https://www.dropbox.com/s/vg0qth94y54hoby/DiplomaThesis_SourceCode.rar?dl=0

Context Manager

```
package com.example.dkotsopoulos.testdiploma;

import android.app.Activity;
import android.content.ComponentName;
import android.content.ContentResolver;
import android.content.ContentUris;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.PackageManager;
import android.content.pm.ResolveInfo;
import android.database.Cursor;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.BitmapShader;
import android.graphics.Canvas;
import android.graphics.Matrix;
import android.graphics.Paint;
import android.graphics.Shader;
import android.graphics.drawable.BitmapDrawable;
import android.graphics.drawable.Drawable;
import android.media.AudioManager;
import android.net.Uri;
import android.os.Bundle;
import android.os.Handler;
import android.preference.PreferenceManager;
import android.provider.ContactsContract;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
import com.google.android.gms.location.Geofence;
import com.google.android.gms.maps.model.LatLng;
import java.io.IOException;
import java.io.InputStream;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Collections;
import java.util.Date;
import java.util.List;
import java.util.Timer;
import java.util.TimerTask;


public class ContextManager extends Activity {
    private static final String TAG_SERVICE = "GroupingLocations";
    private static final String TAG = "InstallShortcuts";
    private static double threshold = 300; // Distance between two points in
METERS
    public static List<GroupPointObject> points = new Ar-
rayList<GroupPointObject>();
    DBHelper dbHelper;
    ArrayList<Geofence> mGeofences;
    ArrayList<LatLng> mGeofenceCoordinates;
    ArrayList<Integer> mGeofenceRadius;
    String[] geocoordinate = new String[2];
    GeofenceStore mGeofenceStore;
    List<ApplicationObject> Newapp;
    CalendarAdapter adapter;
    Button mapbutton,ShowEvents,Directions,StopAlarm,add,remove;
    Timer timer;
    Timer timerservices;
    Timer timerservicesShortcuts;
    TimerTask timerTask;
    TimerTask timerTaskservices;
    TimerTask timerTaskservicesShortcuts;
    final Handler handler = new Handler();
    final Handler handlerservices = new Handler();
    final Handler handlerservicesShortcuts = new Handler();
    String TimersAreRunning;


    @Override
    protected void onCreate(Bundle savedInstanceState) {


        SharedPreferences location = PreferenceManag-
er.getDefaultSharedPreferences(getApplicationContext());
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_context_manager);
        mapbutton = (Button) findViewById(R.id.mapbutton);
        ShowEvents=(Button) findViewById(R.id.ShowEvents);
        Directions = (Button) findViewById(R.id.directionsbutton);
        StopAlarm = (Button) findViewById(R.id.stopalarmServicesbutton);



//////////////////////////////////////////////////////////////////////////////
/////////////////

/*Here we create the clustering for Work centroid and Home centroid*/

        dbHelper = new DBHelper(this);
        points = dbHelper.GetGroupedGPS();
        Log.d(TAG_SERVICE, String.valueOf(points.size()));
        for (int i = 0; i < points.size(); i++) {
            for (int j = i + 1; j < points.size(); ) {
                GroupPointObject pointHere = points.get(i);
                GroupPointObject pointThere = points.get(j);
                double length = distance(pointHere.getY(), pointThere.getY(),
pointHere.getX(), pointThere.getX(), 0.0, 0.0);

                if (length <= threshold) {
                    points.get(i).xCluster = (points.get(i).xCluster +
points.get(j).getX());
                    points.get(i).yCluster = (points.get(i).yCluster +
points.get(j).getY());
```

```
                    points.get(i).cluster_volume++;
                    try {
                        if (check-
time(points.get(j).getdatetime()).equals("Work")) {
                            points.get(i).Work++;}
                        else {points.get(i).Home++;}
                    } catch (ParseException e) {e.printStackTrace();}
                    points.remove(j);
                } else {j += 1;}
            }
        }
        LatLng Home = new LatLng(0, 0);
        LatLng Work = new LatLng(0, 0);
        Collections.sort(points);

        location.edit().clear();
        if (points.size() >= 2) {
            if (points.get(0).Work < points.get(0).Home) {
                //Here i should create the markers for map
                Log.d("Home: ", points.get(0).toString());
                Home = new LatLng(points.get(0).getX(), points.get(0).getY());

            } else {
                //Here i should create the markers for map
                Log.d("Work: ", points.get(0).toString());
                Work = new LatLng(points.get(0).getX(), points.get(0).getY());
            }
            if (points.get(1).Work < points.get(1).Home) {
                //Here i should create the markers for map
                Log.d("Home: ", points.get(1).toString());
                Home = new LatLng(points.get(1).getX(), points.get(1).getY());
            } else {
                //Here i should create the markers for map
                Log.d("Work: ", points.get(1).toString());
                Work = new LatLng(points.get(1).getX(), points.get(1).getY());
            }
        } else {
            Toast.makeText(getApplicationContext(), "Less than two clusters!",
Toast.LENGTH_LONG).show();
        }


        // Writing data a variable and pass them to the map activity for dis-
playing them
        geocoordinate[0] = String.valueOf(Home.latitude) + "," +
String.valueOf(Home.longitude);
        geocoordinate[1] = String.valueOf(Work.latitude) + "," +
String.valueOf(Work.longitude);
        SharedPreferences.Editor editor = location.edit();
        editor.putString("LatHome", String.valueOf(Home.latitude)).commit();
        editor.putString("LogHome", String.valueOf(Home.longitude)).commit();
        editor.putString("LatWork", String.valueOf(Work.latitude)).commit();
        editor.putString("LogWork", String.valueOf(Work.longitude)).commit();
        mapbutton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Log.d(TAG_SERVICE, "onClick: starting ContextManager");
                startActivity(new Intent(ContextManager.this, MapActivi-
ty.class).putExtra("Geolocations", geocoordinate));
            }
        });
/********************************************End of clustering creation
***********************/

////////////////////////////////////////////////////////////////////////////
////////////////

/*********************Here we implement the geofenc-
```

```
ing*****************************************/
        // Initializing variables
        mGeofences = new ArrayList<Geofence>();
        mGeofenceCoordinates = new ArrayList<LatLng>();
        mGeofenceRadius = new ArrayList<Integer>();
        mGeofenceRadius.add(100); //Radious in meters
        mGeofenceCoordinates.add(Work);
        mGeofenceCoordinates.add(Home);
        // Home, the coordinates of the center of the geofence and the radius
in meters.
        mGeofences.add(new
Geofence.Builder().setRequestId("Home").setCircularRegion(mGeofenceCoordinates
.get(1).latitude, mGeofenceCoordinates.get(1).longitude,
                mGeofenceRadius.get(0).intValue())

.setExpirationDuration(Geofence.NEVER_EXPIRE).setLoiteringDelay(3600000)
                .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER |
Geofence.GEOFENCE_TRANSITION_DWELL |
Geofence.GEOFENCE_TRANSITION_EXIT).build());

        // Work, the coordinates of the center of the geofence and the radius
in meters.
        mGeofences.add(new Geofence.Builder().setRequestId("Work")
                .setCircularRegion(mGeofenceCoordinates.get(0).latitude,
mGeofenceCoordinates.get(0).longitude, mGeofenceRadius.get(0).intValue())
                .setExpirationDuration(Geofence.NEVER_EXPIRE)
                        // Required when we use the transition type of
GEOFENCE_TRANSITION_DWELL
                .setLoiteringDelay(3600000)// 60minutes inside the geofence an
alert will be sent
                .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER |
Geofence.GEOFENCE_TRANSITION_DWELL |
Geofence.GEOFENCE_TRANSITION_EXIT).build());

        // Add the geofences to the GeofenceStore object.
        mGeofenceStore = new GeofenceStore(this, mGeofences);

/**************End of geofenc-
ing************************************************************ */

//////////////////////////////////////////////////////////////////////////////
/////////////////

/************************Here we implement mute after 23:59
event*********************************/
        SharedPreferences shareref = PreferenceManag-
er.getDefaultSharedPreferences(getApplicationContext());
        TimersAreRunning= shareref.getString("TimerOn", "");
        if (TimersAreRunning.equals("True"))
        {
            try {
                startTimerServices();
            } catch (Exception e) {
                e.printStackTrace();
            }
            /*************************End  mute after 23:59
event********************************************/

            /************************Here we implement shortcuts
rule*********************************/
            // 3 most used apps
            try {
                startTimer();
                //whatsapp, hangouts, top three callers
                startTimerShortcuts();

            } catch (ParseException e) {
                e.printStackTrace();
            }
        }


/**************************End  shortcuts
**********************************************/
////////////////////////////////////////////////////////////////////////////////
/////////////////

/**********************Here we implement the selection of shortcuts manual-
ly*****************/


    add = (Button) findViewById(R.id.buttonAddShortcut);
    add.setOnClickListener(new View.OnClickListener() {

        @Override
        public void onClick(View v) {
            //Add shortcut on Home screen
            addShortcut();
            AddContactShortcut();
            AddWhatsShortcuts();
            AddSMSShortcuts();

        }
    });

    //Add listener to remove shortcut button
    remove = (Button) findViewById(R.id.buttonRemoveShortcut);
    remove.setOnClickListener(new View.OnClickListener() {

        @Override
        public void onClick(View v) {

            //Remove shortcut from Home screen
            removeShortcut();
            RemoveContactShortcut();
            RemoveSMSShortcuts();
            RemoveWhatsShortcuts();
        }
    });


/*********************************End selection of the
shortcuts************************/

////////////////////////////////////////////////////////////////////////////////
/////////////////


/* *****************************Show all the events in your ar-
ea****************************************/
        ShowEvents.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                startActivity(new Intent(ContextManager.this, EventfullActivi-
ty.class));
            }
        });

/*********************End of showing all the events in your ar-
ea***************************************/

////////////////////////////////////////////////////////////////////////////////
/////////////////


/* *****************************Show Direction for mobili-
```

```
ty*****************************************/
        Directions.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v)
            {
                startActivity(new Intent(ContextManager.this,
ChooseTransportMode.class));
            }
        });

/**********************End of showing Direction for mobili-
ty*****************************************/

//////////////////////////////////////////////////////////////////////////////
/////////////////

/* *****************************Stop Tim-
ers!*****************************************/

        StopAlarm.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View v) {
               TimersAreRunning="false";
               stoptimertask();
               stoptimertaskServices();
               stoptimerShortcuts();


           }
       });
/**********************End of Timer for weekly most used applications mod-
ule*****************************************/
    }



/*******************************************************End OnCreate
*******************************************************/


///*******************************************************Start OnPause
*******************************************************/
//    @Override
//    protected void onPause() {
//
//        super.onPause();
//    }
///*******************************************************End OnPause
*******************************************************/
//
///*******************************************************Start OnResume
*******************************************************/
//    @Override
//    protected void onResume() {

//        super.onResume();
//    }
//
///*******************************************************End OnResume
*******************************************************/



    /*************************************Assisting func-
tions****************************/
```

```
//////////////////////Here we are checking the
time/////////////////////////////////
    public static String checktime(Long time) throws ParseException {
        //Checking whether the timestamp is on home or work based on fixed
values. (We exclude Weekends because they affect the reliability of the sam-
ple)
        //Start Time
        Date inTime = new SimpleDateFormat("HH:mm:ss").parse("09:00:00");
        Calendar calendar1 = Calendar.getInstance();
        calendar1.setTime(inTime);

        //End Time
        Date finTime = new SimpleDateFormat("HH:mm:ss").parse("18:00:00");
        Calendar calendar2 = Calendar.getInstance();
        calendar2.setTime(finTime);
        //Current Time
        String timeLog2 = new SimpleDateFormat("EEEE").format(time);
        String timeLog = new SimpleDateFormat("HH:mm:ss").format(time);
        Date checkTime = new SimpleDateFormat("HH:mm:ss").parse(timeLog);
        Calendar calendar3 = Calendar.getInstance();
        calendar3.setTime(checkTime);
        Date actualTime = calendar3.getTime();
        if (timeLog2.matches("Saturday|Sunday")) {
//            Log.d(TAG_SERVICE, "Nothing");
            return "Nothing";
        } else {

            if (actualTime.after(calendar1.getTime()) && actual-
Time.before(calendar2.getTime())) {
//                Log.d(TAG_SERVICE, "Work");
                return "Work";
            } else {
//                Log.d(TAG_SERVICE, "Home");
                return "Home";
            }

        }
    }

//////////////////////Here we are checking the distance between two
points////////////////////////
    public static double distance(double lat1, double lat2, double lon1,
                                  double lon2, double el1, double el2) {

        final int R = 6371; // Radius of the earth
        //Wg84 and this is a function that calculates coordinates to meters
        Double latDistance = Math.toRadians(lat2 - lat1);
        Double lonDistance = Math.toRadians(lon2 - lon1);
        Double a = Math.sin(latDistance / 2) * Math.sin(latDistance / 2)
                + Math.cos(Math.toRadians(lat1)) *
Math.cos(Math.toRadians(lat2))
                * Math.sin(lonDistance / 2) * Math.sin(lonDistance / 2);
        Double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
        double distance = R * c * 1000; // convert to meters
        double height = el1 - el2;
        distance = Math.pow(distance, 2) + Math.pow(height, 2);
        return Math.sqrt(distance);
    }

//////////////////////////////Add delete shortcut applica-
tions/////////////////////////////
    public void addShortcut()
    {
        Log.d(TAG, "Adding Shorcuts");
        //Adding shortcut for MainActivity
        //on Home screen

        Intent mainIntent = new Intent(Intent.ACTION_MAIN, null);
        mainIntent.addCategory(Intent.CATEGORY_LAUNCHER);

        final PackageManager pm = getPackageManager();
        List<ResolveInfo> appList = pm.queryIntentActivities(mainIntent, 0);
        Collections.sort(appList, new ResolveInfo.DisplayNameComparator(pm));
        Newapp=dbHelper.GetAllApplications();
    if(!Newapp.equals(null))
    {
        for (ResolveInfo temp : appList) {
            for (int i = 0; i < 3; i++) {
                if
(temp.activityInfo.packageName.equals(Newapp.get(i).getApplicationPackageName(
))) {

                    Intent shortcut = new In-
tent("com.android.launcher.action.INSTALL_SHORTCUT");

                    // Shortcut name
                    shortcut.putExtra(Intent.EXTRA_SHORTCUT_NAME,
temp.loadLabel(pm).toString());
                    shortcut.putExtra("duplicate", true);  // Just create once

                    // Setup current activity shoud be shortcut object
                    ComponentName comp = new Component-
Name(temp.activityInfo.packageName, temp.activityInfo.name);
                    shortcut.putExtra(Intent.EXTRA_SHORTCUT_INTENT, new In-
tent(Intent.ACTION_MAIN).setComponent(comp));
                    Drawable d = temp.loadIcon(pm);
                    Bitmap bitmap = ((BitmapDrawable) d).getBitmap();
                    shortcut.putExtra(Intent.EXTRA_SHORTCUT_ICON, bitmap);
                    sendBroadcast(shortcut);
                }
            }

        }
    }
    }
    public  void removeShortcut() {

        Log.d(TAG, "Deleting Shorcuts");
        //Deleting shortcut for MainActivity on Home screen
        Intent mainIntent = new Intent(Intent.ACTION_MAIN, null);
        mainIntent.addCategory(Intent.CATEGORY_LAUNCHER);

        final PackageManager pm = getPackageManager();
        List<ResolveInfo> appList = pm.queryIntentActivities(mainIntent, 0);
        Collections.sort(appList, new ResolveInfo.DisplayNameComparator(pm));
        Newapp = dbHelper.GetAllApplications();
        if (!Newapp.equals(null)) {
            for (ResolveInfo temp : appList) {
                for (int i = 0; i < 3; i++) {
                    if
(temp.activityInfo.packageName.equals(Newapp.get(i).getApplicationPackageName(
))) {


                        Intent shortcut = new In-
tent("com.android.launcher.action.UNINSTALL_SHORTCUT");
                        shortcut.putExtra(Intent.EXTRA_SHORTCUT_NAME,
temp.loadLabel(pm).toString());
                        String appClass = temp.activityInfo.packageName + "."
+ temp.activityInfo.name;
                        ComponentName comp = new Component-
Name(temp.activityInfo.packageName, appClass);
                        shortcut.putExtra(Intent.EXTRA_SHORTCUT_INTENT, new
Intent(Intent.ACTION_MAIN).setComponent(comp));
                        sendBroadcast(shortcut);
                        }
                    }
                }
            }
        }

/////////////////////////////////// Add delete contacts
shortuts/////////////////////////////////
    private void AddContactShortcut() {
        Log.d(TAG, "Adding Contact Shorcuts");
        DBHelper dbHelper= new DBHelper(this);

        List<CallObject> tester= new ArrayList<CallObject>();
        String checker= null;
        try {
            checker = checktime(System.currentTimeMillis());
        } catch (ParseException e) {
            e.printStackTrace();
        }
        if (checker.equals("Work"))
        {
            tester= dbHelper.GetTopThreeCallers();
        }
        else{
            tester= dbHelper.GetTopThreeCallersAfterWork();
        }

        for (int i=0; i<tester.size();i++) {

            if (!tester.get(i).getcallname().equals("Unknown")) {

                Log.d(TAG, tester.get(i).getcallname());
                Log.d(TAG, tester.get(i).getphnumber().toString());
                Intent shortCutInt = new  Intent(Intent.ACTION_DIAL);
                shortCutInt.setData(Uri.parse("tel:" + test-
er.get(i).getphnumber().toString()));
                shortCutInt.putExtra("duplicate", true);  // Just create once
                Intent addInt = new Intent();
                addInt.putExtra(Intent.EXTRA_SHORTCUT_INTENT, shortCutInt);
                addInt.putExtra(Intent.EXTRA_SHORTCUT_NAME, test-
er.get(i).getcallname().toString());
                Bitmap bitmap = retrieveContactPhoto(getApplicationContext(),
tester.get(i).getphnumber().toString());
                Drawable myDrawable =
getResources().getDrawable(R.drawable.phone);
                Bitmap myLogo = ((BitmapDrawable) myDrawable).getBitmap();
                if (bitmap == null) {
                    addInt.putExtra(Intent.EXTRA_SHORTCUT_ICON_RESOURCE,
                            Intent.ShortcutIconResource.fromContext(this,
R.drawable.phone));
                } else {
                    addInt.putExtra(Intent.EXTRA_SHORTCUT_ICON, over-
lay(myLogo,bitmap));
                }

addInt.setAction("com.android.launcher.action.INSTALL_SHORTCUT");
                sendBroadcast(addInt);

            }
        }
    }
    private void RemoveContactShortcut(){
        DBHelper dbHelper= new DBHelper(this);
        List<CallObject> tester= new ArrayList<CallObject>();
        tester= dbHelper.GetTopThreeCallers();
        for (int i=0; i<tester.size();i++) {
            if (!tester.get(i).getcallname().equals("Unknown"))
            {
                Intent shortCutInt = new  Intent(Intent.ACTION_DIAL);
                shortCutInt.setData(Uri.parse("tel:" + test-
er.get(i).getphnumber().toString()));
                Intent removeshortcut = new Intent();
                removeshortcut.putExtra(Intent.EXTRA_SHORTCUT_INTENT,
shortCutInt);
                removeshortcut.putExtra(Intent.EXTRA_SHORTCUT_NAME, test-
er.get(i).getcallname().toString());
                re-
moveshortcut.setAction("com.android.launcher.action.UNINSTALL_SHORTCUT");
                sendBroadcast(removeshortcut);

            }
        }

    }

//////////////////////////////// Add delete whatup
shortuts//////////////////////////////
    private void AddWhatsShortcuts(){
        Log.d(TAG, "Adding WhatsApp Shorcuts");
        DBHelper dbHelper= new DBHelper(this);
        List<CallObject> tester= new ArrayList<CallObject>();
        tester= dbHelper.GetTopThreeCallers();
        Drawable AppIcon = null;
        boolean installed = appInstalledOrNot("com.whatsapp");
        if (installed) {
            try {
                AppIcon = getPackageManag-
er().getApplicationIcon("com.whatsapp");
            } catch (PackageManager.NameNotFoundException e) {
                e.printStackTrace();
            }
            Bitmap Appbitmap = ((BitmapDrawable) AppIcon).getBitmap();
            for (int i = 0; i < tester.size(); i++) {

                if (!tester.get(i).getcallname().equals("Unknown")) {
                    Cursor c = getContentResolv-
er().query(ContactsContract.Data.CONTENT_URI,
                            new String[]{ContactsContract.Contacts.Data._ID},
ContactsContract.Data.DATA1 + "=?",
                            new String[]{tester.get(i).getphnumber() +
"@s.whatsapp.net"}, null);
                    c.moveToFirst();
                    if (c.getCount() != 0) {
                        Intent whatupsintent = new Intent(Intent.ACTION_VIEW,
Uri.parse("content://com.android.contacts/data/" + c.getString(0)));
                        Intent addInt = new Intent();
                        addInt.putExtra("duplicate", false);
                        addInt.putExtra(Intent.EXTRA_SHORTCUT_INTENT, whatup-
sintent);
                        addInt.putExtra(Intent.EXTRA_SHORTCUT_NAME, test-
er.get(i).getcallname().toString());
                        Bitmap bitmap = retrieveContactPho-
to(getApplicationContext(), tester.get(i).getphnumber().toString());
                        if (bitmap == null) {
                            addInt.putExtra(Intent.EXTRA_SHORTCUT_ICON, Ap-
pbitmap);
                        } else {
                            addInt.putExtra(Intent.EXTRA_SHORTCUT_ICON, over-
lay(Appbitmap, bitmap));
                        }

addInt.setAction("com.android.launcher.action.INSTALL_SHORTCUT");
                        sendBroadcast(addInt);
                    }
                    c.close();
                }
            }
        }
    }
    private void RemoveWhatsShortcuts(){

        DBHelper dbHelper= new DBHelper(this);
        List<CallObject> tester= new ArrayList<CallObject>();
        tester= dbHelper.GetAllCalls();
        Drawable AppIcon = null;
        try {
            AppIcon = getPackageManager().getApplicationIcon("com.whatsapp");
        } catch (PackageManager.NameNotFoundException e) {
            e.printStackTrace();
        }
        Bitmap Appbitmap = ((BitmapDrawable)AppIcon).getBitmap();
        for (int i=0; i<tester.size();i++) {

            if (!tester.get(i).getcallname().equals("Unknown"))
            {
                Cursor c = getContentResolv-
er().query(ContactsContract.Data.CONTENT_URI,
                        new String[]{ContactsContract.Contacts.Data._ID}, Con-
tactsContract.Data.DATA1 + "=?",
                        new String[]{tester.get(i).getphnumber() +
"@s.whatsapp.net"}, null);
                c.moveToFirst();
                if (c.getCount()!=0)
                {
                    Intent whatupsintent = new Intent(Intent.ACTION_VIEW,
Uri.parse("content://com.android.contacts/data/" + c.getString(0)));
                    Intent addInt = new Intent();
                    addInt.putExtra(Intent.EXTRA_SHORTCUT_INTENT, whatupsin-
tent);
                    addInt.putExtra(Intent.EXTRA_SHORTCUT_NAME, test-
er.get(i).getcallname().toString());
                    Bitmap bitmap = retrieveContactPho-
to(getApplicationContext(), tester.get(i).getphnumber().toString());
                    if (bitmap == null) {
                        addInt.putExtra(Intent.EXTRA_SHORTCUT_ICON, Appbit-
map);
                    } else {
                        addInt.putExtra(Intent.EXTRA_SHORTCUT_ICON, over-
lay(Appbitmap,bitmap));
                    }

addInt.setAction("com.android.launcher.action.UNINSTALL_SHORTCUT");
                    sendBroadcast(addInt);

                }
                c.close();
            }
        }
    }

//////////////////////////////// Add delete Hangouts
shortuts/////////////////////////////
    private void AddSMSShortcuts() {
        Log.d(TAG, "Adding SMS Shorcuts");
        DBHelper dbHelper = new DBHelper(this);
        List<SmsObject> tester = new ArrayList<SmsObject>();
        tester = dbHelper.GetMostThreeSMS();
        Drawable AppIcon = null;
        boolean installed = appInstalledOrNot("com.google.android.talk");
        if (installed) {
            try {
                AppIcon = getPackageManag-
er().getApplicationIcon("com.google.android.talk");
            } catch (PackageManager.NameNotFoundException e) {
                e.printStackTrace();
            }

            Bitmap Appbitmap = ((BitmapDrawable) AppIcon).getBitmap();

            for (int i = 0; i < tester.size(); i++) {

                if (!tester.get(i).getPerson().equals("Unknown")) {

                    Intent smsIntent = new Intent(Intent.ACTION_VIEW);
                    smsIntent.setType("vnd.android-dir/mms-sms");
                    smsIntent.setData(Uri.parse("sms:" + test-
er.get(i).getaddress()));
                    Intent addInt = new Intent();
                    addInt.putExtra("duplicate", true);
                    addInt.putExtra(Intent.EXTRA_SHORTCUT_INTENT, smsIntent);
                    addInt.putExtra(Intent.EXTRA_SHORTCUT_NAME, test-
er.get(i).getPerson().toString());
                    Bitmap bitmap = retrieveContactPho-
to(getApplicationContext(), tester.get(i).getaddress().toString());
                    if (bitmap == null) {
                        addInt.putExtra(Intent.EXTRA_SHORTCUT_ICON, Appbit-
map);
                    } else {
                        addInt.putExtra(Intent.EXTRA_SHORTCUT_ICON, over-
lay(Appbitmap, bitmap));
                    }

addInt.setAction("com.android.launcher.action.INSTALL_SHORTCUT");
                    sendBroadcast(addInt);

                }
            }
        }
    }
    private void RemoveSMSShortcuts() {

        DBHelper dbHelper= new DBHelper(this);
        List<CallObject> tester= new ArrayList<CallObject>();
        tester= dbHelper.GetTopThreeCallers();
        Drawable AppIcon = null;
        boolean installed = appInstalledOrNot("com.google.android.talk");
        if (installed) {
            try {
                AppIcon = getPackageManag-
er().getApplicationIcon("com.google.android.talk");
            } catch (PackageManager.NameNotFoundException e) {
                e.printStackTrace();
            }
            Bitmap Appbitmap = ((BitmapDrawable) AppIcon).getBitmap();

            for (int i = 0; i < tester.size(); i++) {

                if (!tester.get(i).getcallname().equals("Unknown")) {

                    Intent smsIntent = new Intent(Intent.ACTION_VIEW);
                    smsIntent.setType("vnd.android-dir/mms-sms");
                    smsIntent.setData(Uri.parse("sms:" + test-
er.get(i).getphnumber()));
                    Intent addInt = new Intent();
                    addInt.putExtra(Intent.EXTRA_SHORTCUT_INTENT, smsIntent);
                    addInt.putExtra(Intent.EXTRA_SHORTCUT_NAME, test-
er.get(i).getcallname().toString());
                    Bitmap bitmap = retrieveContactPho-
to(getApplicationContext(), tester.get(i).getphnumber().toString());
```

```
                    if (bitmap == null) {
                        addInt.putExtra(Intent.EXTRA_SHORTCUT_ICON, Appbit-
map);
                    } else {
                        addInt.putExtra(Intent.EXTRA_SHORTCUT_ICON, over-
lay(Appbitmap, bitmap));
                    }

addInt.setAction("com.android.launcher.action.UNINSTALL_SHORTCUT");
                    sendBroadcast(addInt);

                }
            }
        }
    }


////////////////////////////////Find the contact photo from the call num-
ber/////////////////////////
    public static Bitmap retrieveContactPhoto(Context context, String number)
{
        ContentResolver contentResolver = context.getContentResolver();
        String contactId = null;
        Uri uri =
Uri.withAppendedPath(ContactsContract.PhoneLookup.CONTENT_FILTER_URI,
Uri.encode(number));

        String[] projection = new
String[]{ContactsContract.PhoneLookup.DISPLAY_NAME, ContactsCon-
tract.PhoneLookup._ID};

        Cursor cursor =
                contentResolver.query(
                        uri,
                        projection,
                        null,
                        null,
                        null);

        if (cursor != null) {
            while (cursor.moveToNext()) {
                contactId = cur-
sor.getString(cursor.getColumnIndexOrThrow(ContactsContract.PhoneLookup._ID));
            }
            cursor.close();
        }

        Bitmap photo = null;

        try {
            InputStream inputStream = ContactsCon-
tract.Contacts.openContactPhotoInputStream(context.getContentResolver(),
                    ContentU-
ris.withAppendedId(ContactsContract.Contacts.CONTENT_URI, new
Long(contactId)));

            if (inputStream != null) {
                photo = BitmapFactory.decodeStream(inputStream);
                assert inputStream != null;
                inputStream.close();
            }else
            {
                photo=null;
            }



        } catch (IOException e) {
            e.printStackTrace();
        }

        if (photo!=null) {
            Bitmap circleBitmap = photo.createBitmap(photo.getWidth(), pho-
to.getHeight(), Bitmap.Config.ARGB_8888);

            BitmapShader shader = new BitmapShader(photo,
Shader.TileMode.CLAMP, Shader.TileMode.CLAMP);
            Paint paint = new Paint();
            paint.setShader(shader);
            paint.setAntiAlias(true);
            Canvas c = new Canvas(circleBitmap);
            c.drawCircle(photo.getWidth() / 2, photo.getHeight() / 2, pho-
to.getWidth() / 2, paint);
            return circleBitmap;



        }
        else
            return photo;
    }

    // Merge the application icon with the contact photo
    public static Bitmap overlay(Bitmap bmp1, Bitmap bmp2) {
        Bitmap bmOverlay = Bitmap.createBitmap(bmp1.getWidth(),
bmp1.getHeight(), bmp1.getConfig());
        Canvas canvas = new Canvas(bmOverlay);
        canvas.drawBitmap(bmp1, new Matrix(), null);
        canvas.drawBitmap(bmp2, 0, 0, null);
        return bmOverlay;
    }

////////Timer for weekly repeated applications' shortcuts install-
ment.////////////////////////////
    public void startTimer() throws ParseException {
        // get start of this week in milliseconds
        Calendar cal = Calendar.getInstance();
        cal.set(Calendar.HOUR_OF_DAY, 0); // ! clear would not reset the hour
of day !
        cal.clear(Calendar.MINUTE);
        cal.clear(Calendar.SECOND);
        cal.clear(Calendar.MILLISECOND);
        cal.set(Calendar.DAY_OF_WEEK, cal.getFirstDayOfWeek());
        Log.d("Start of this week:       ", cal.getTime().toString());
        Date finaldate = cal.getTime();
        int interval = 1000 * 60 *10080;//1440;// 24 hour
        //set a new Timer
        timer = new Timer();
        //initialize the TimerTask's job
        initializeTimerTask();
        //schedule the timer, after the first 10sec the TimerTask will run
every 24 hour
        timer.schedule(timerTask, finaldate, interval); //
    }

    public void stoptimertask() {

        //stop the timer, if it's not already null
        Log.d(TAG, "Stopping timer task");
        if (timer != null)
        {
            timer.cancel();
            timer = null;
        }

    }
    public void initializeTimerTask() {
        Log.d(TAG, "initializeTimerTask");
        timerTask = new TimerTask() {
            public void run() {
                //use a handler to run a toast that shows the current
timestamp

                handler.post(new Runnable() {

                    public void run()
                    {
                        removeShortcut();
                        addShortcut();

                    }

                });

            }

        };

    }

//////////////////////////////////////////////////////////////////////////////
////////////////////
   // Check if user is home and the time is 23:57 and mute phone

    public void startTimerServices()
    {
    //set a new Timer
        timerservices = new Timer();
        //initialize the TimerTask's job
        initializeTimerTaskServices();
        //schedule the timer, after the first 10sec the TimerTask will run
every 2,5min
        timerservices.schedule(timerTaskservices, 10000,150000); //

    }
    public  void stoptimertaskServices()
    {

        Log.d(TAG, "Stopping timer task Services");
        //stop the timer, if it's not already null
        if (timerservices != null)
        {
            timerservices.cancel();
            timerservices = null;
        }

    }
    public void initializeTimerTaskServices() {

        timerTaskservices = new TimerTask() {
            public void run() {
                //use a handler to run a toast that shows the current
timestamp

                handlerservices.post(new Runnable() {

                    public void run()
                    {
                        //Calendar Service
                        startService(new Intent(ContextManager.this, Calendar-
Listener.class));

                        //CalendarEvents Service
                        startService(new Intent(ContextManager.this, Calendar-
Service.class));
                        ///Application Service
                        startService(new Intent(ContextManager.this, Applica-
tionListener.class));
                        /*********************Rule number 2- Check if user is
home and midnight*****************************************/

                        SharedPreferences check = PreferenceManag-
er.getDefaultSharedPreferences(getApplicationContext());
                        String Checker= check.getString("CheckHome", "");
                        Calendar currTime = Calendar.getInstance();
                        Log.d("ContextManager Outside",Checker);
                        int hour = currTime.get(Calendar.HOUR_OF_DAY);
                        int minute = currTime.get(Calendar.MINUTE);
                        // 0 ==work ,1== home
                        if ((Checker.equals("1")) && (hour==23) && (mi-
nute>57))
                        {
                            /**********************************Here we imple-
ment the walking service************************/
                            stopService(new Intent(ContextManager.this, Walk-
ingService.class));
                            /**********************************End of the
walking service************************/

                            AudioManager mAudioManager = (AudioManager) get-
SystemService(Context.AUDIO_SERVICE);
                            mAudioManag-
er.setVibrateSetting(AudioManager.VIBRATE_TYPE_RINGER, AudioManag-
er.VIBRATE_SETTING_OFF);
                            mAudioManag-
er.setRingerMode(AudioManager.RINGER_MODE_SILENT);

                            Log.d("ContextManager Inside",Checker);
                        }

                        /*********************End of Rule number 2- Check if
user is home and midnight*****************************************/

                    }

                });

            }

        };

    }
/////////////////////////////////////////////////////////////////////////////
////////////////////
// For adding or removing contact shortcuts


    public void startTimerShortcuts() throws ParseException {
        Log.d(TAG, "Entering timer task Shortcuts");
        //set a new Timer
        timerservicesShortcuts = new Timer();
        //initialize the TimerTask's job
        initializeTimerShortcuts();
        //schedule the timer, after the first 10sec the TimerTask will run
every 2,5min
        timerservicesShortcuts.schedule(timerTaskservicesShortcuts,
10000,150000); //
}

    public void stoptimerShortcuts() {
        //stop the timer, if it's not already null
        if (timerservicesShortcuts != null)
        {
            timerservicesShortcuts.cancel();
            timerservicesShortcuts = null;
        }

    }

    public void initializeTimerShortcuts() {

        timerTaskservicesShortcuts = new TimerTask() {
            public void run() {
                //use a handler to run a toast that shows the current
timestamp
                Log.d(TAG, "initializeTimerShortcuts");
                handlerservicesShortcuts.post(new Runnable() {

                    public void run()
                    {
                        try {
                            int check-
er=checkcurrenttime(System.currentTimeMillis());
                            if (checker==1) {
                                AddContactShortcut();
                                AddSMSShortcuts();
                                AddWhatsShortcuts();
                            }
                        } catch (ParseException e) {
                            e.printStackTrace();
                        }


                    }

                });

            }

        };

    }



////////////////////////////////////////////////////////////////////////////////
////////////////////

    public static int checkcurrenttime(Long time) throws ParseException {

        //End Time
        Date inTime = new SimpleDateFormat("HH:mm:ss").parse("17:15:00");
        Calendar calendar1 = Calendar.getInstance();
        calendar1.setTime(inTime);

        //Current Time
        String timeLog2 = new SimpleDateFormat("EEEE").format(time);
        String timeLog = new SimpleDateFormat("HH:mm:ss").format(time);
        Date checkTime = new SimpleDateFormat("HH:mm:ss").parse(timeLog);
        Calendar calendar3 = Calendar.getInstance();
        calendar3.setTime(checkTime);
        Date actualTime = calendar3.getTime();
        if (actualTime.after(calendar1.getTime()) ) {return 1;} else {return
0;}
    }


////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////
    private boolean appInstalledOrNot(String uri) {
        PackageManager pm = getPackageManager();
        boolean app_installed;
        try {
            pm.getPackageInfo(uri, PackageManager.GET_ACTIVITIES);
            app_installed = true;
        }
        catch (PackageManager.NameNotFoundException e) {
            app_installed = false;
        }
        return app_installed;
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is pre-
sent.
        getMenuInflater().inflate(R.menu.menu_context_manager, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }


}
```