CHAIR OF WIRELESS COMMUNICATIONS
FACULTY OF ELECTRONICS & TELECOMMUNICATIONS
POZNAŃ UNIVERSITY OF TECHNOLOGY

Bachelor of Science Thesis

APPLICATION FOR ANDROID-BASED SMARTPHONE OFFERING SELECTED ECALL SERVICES

by

Maciej Piotrowski

Supervisor:
Andrzej Stelter, PhD

Poznań, 2012

Abstract

The objective of this thesis was to create mobile application for Android-based smartphone. The program offers selected services of eCall system. In the thesis, the author included smartphone market and mobile operating systems description. The eCall system concept is introduced as well. The author described implemented life-saving application for Android platform and its main components, wireless interfaces with their usage and SSID sensing concept.

Celem niniejszej pracy było stworzenie aplikacji mobilnej dla terminali typu smartfon z systemem operacyjnym Android. Napisana aplikacja oferuje wybrane usługi systemu eCall. W pracy autor zamieścił podstawowe informacje na temat rynku terminali typu smartfon oraz systemów operacyjnych dla urządzeń mobilnych, przedstawiona została także koncepcja systemu eCall. Autor zawarł w pracy opis stworzonego programu ratunkowego oraz jego poszczególnych komponentów. Przedstawione zostały możliwości wykorzystania interfejsów bezprzewodowych na platformie Android oraz opis koncepcji wykrywania SSID.

Contents

Chapter 1

Introduction

Alarming public services is always an issue, especially when a person does not know how and who to call for help, or when he or she is confused about his or her current location. But when something goes wrong, the time is the most crucial value in life-saving.

In the daylight of growing smartphone market and the powerful application ecosystem, a comprehensive program for alarming can be created. Geographic coordinates are the universal language that everyone can speak and understand. It is also a reliable information about one's position in case of an accident. The smartphone is an entity, which can obtain user's position very fast; either can it serve to call police, paramedics or firefighters.

European Union (EU) eCall concept is an alarming system designed for automotive sector. This BSc thesis goal is to adapt selected eCall services to many environments: city, village, forest, mountains, seaside and every place where cellular network connectivity can be established. The main entity should be a smartphone terminal. The implementation of a worldwide applicable solution for Android-based smarpthones is described further in this thesis. The application is designed to provide a fast and intelligent interface between the Search and Rescue (SAR) services and a victim. An extension of wireless interfaces usage inside life-saving application is also discussed.

The author motivates his work mainly by his interests in Android application development and programming. Thesis subject is also influenced by winning European Satellite Navigation Competition (ESNC) 2011 Bavaria prize by Maciej Piotrowski and his friend Michał Wojtysiak [1]. The winning project - Your Entertainment & Tracking Interface (YETI) - is an eCall concept-based system for skiers and snowboarders. It provides SAR along with entertainment services, where main entity is user's smartphone.

In the Chair of Wireless Communications currently two BSc theses are carried out on *eCall system adaptation* subject. Michał Wojtysiak for his thesis developed life-saving application for Windows Phone platform. Due to the same topic of both theses some of *Chapter 2* content is common for both papers. Michał Wojtysiak's contribution to this work is creating the emergency numbers database file (described in *Chapter 3.4.1*) and eCall section in *Chapter 2.2*, as well as designing some graphics for application. Everything else contained in this thesis is author's individual contribution.

The thesis is divided in five chapters. The First one is an introduction to the work. Second one contains information about smartphone market and eCall system fundamentals. The third chapter depicts basics of programs for Android operating system and describes the life-saving application. Chapter number four introduces wireless interfaces, their usage in Android-based smartphone and *SSID sensing* concept. Last chapter is a summary of this thesis.

The author would like to thank: Andrzej Stelter, PhD, for the supervision and tips how to proceed with work; Michał Wojtysiak for his contribution to the project; Piotr Koczorowski for spell checking and proof reading; Kamil Grzegorek, Bartosz Kaczmarek and Agata Szyndlarewicz for testing and feedback on the life-saving application on Android-based smartphone.

Chapter 2

Theoretical information

2.1 Smartphones

2.1.1 Introduction

A smartphone is an intelligent cellular phone. Apart from casual functions such as performing calls and sending text messages it also contains a photo camera and usually a media player, but moreover it also has functionalities of a Personal Digital Assistant (PDA). This section of thesis introduces market trends in telecommunications area with high emphasis on smartphones, the operating systems that run on them, their functions and their future capabilities.

2.1.2 Market trends

At the end of 2009 there were up to 4.6 billion mobile subscriptions on Earth. Throughout 2 years, this number has increased to 5.3 billion and is still growing. This means that 77% of the world population uses mobile phones [2].

Mostly used devices are still feature phones, but smartphone penetration increases year by year and shows the strongest growth in mobile devices sales. When cheap feature phones are sold in less-developed countries, smartphone manufacturers found their customers mainly in higly-developed parts of the world.

Ericsson Vice President, Ulf Ewaldsson, forecasts that by the end of year 2016 the mobile subscriptions increase will reach the peak of 8.5 billion active Subscriber Identity Module (SIM) cards and half of them will be installed in smartphones. This forecast slowly becomes truth - last year, the number of shipped cellular phones was around 1.4 billion items, with 302 million smartphones sold. Main smartphone markets belong to USA and west Europe.

Figure 2.1 and corresponding table 2.1 shows top five mobile phone manufacturers market share by 2010 global sales. The strongest position is held by Nokia (453 million

devices sold), Samsung (280.2 million devices sold) and LG (116.7 million devices sold) with 32.63%, 20.18% and 8.41% market share respectively. These results are not surprising since Nokia, Samsung and LG have been producing handsets for many years focusing also on cheap feature phones for customers from non-rich countries. Another very successful producers were ZTE and Research In Motion (RIM) with 3.73% market share with 51.8 million phones sold and 3.52% market share with 48.8 million devices sold respectively. The key point, which decided about company incomes and demand for its goods, was average price for handset.

Table 2.1: Top five mobile phone manufacturers by 2010 global sales (in millions) [according to Gartner]

Manufacturer	Sales	Market share
Nokia	453	32,63%
Samsung	280.2	20,18%
LG	116.7	8,41%
ZTE	51.8	3,73%
RIM	48.8	3,52%
Others	437.7	31,53%
Total	1 388.2	100%

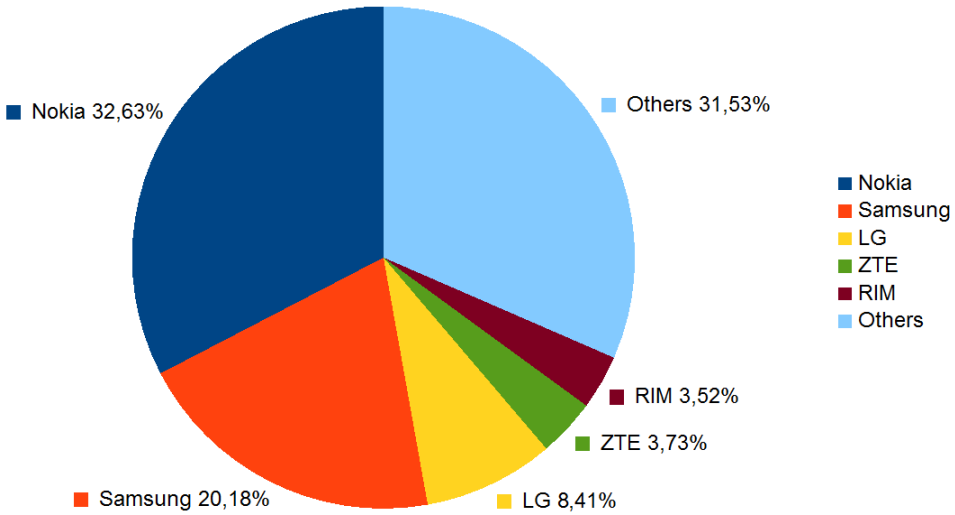

Figure 2.1: Top five mobile phone manufacturers by 2010 global sales [according to Gartner]

The production of the first smartphone gradually began the era of intelligent cellulars with extended functionalities. For many producers it is not convenient to build their revenue model on selling cheap mobile phones. Companies such as Apple, HTC and RIM base their incomes only on manufacturing smartphones and providing

many services for their users as added value. Also in smartphone target market the decision which phone brand to choose was made based on price.

In the figure 2.2 and from the corresponding table 2.2 top five smartphone manufacturers in 2010 are displayed. The most successful in this area, as in previous ranking, was Nokia - 33.15% of market belongs to company from Finland with 100.3 million devices sold. Second most successful company was RIM with their BlackBerry smartphone - 16.13% market share and 48.8 million phones sold. Another very successful company was Apple with iPhone product - 15.7% of market share and 47.5 million cellulars sold. According to data from table 2.2 the fourth and fifth best selling manufacturer were Samsung (7.6% of market with 23 million cellulars sold) and HTC (7.11% of market with 21.5 million handsets sold) respectively.

Table 2.2: Top five smartphone manufacturers by 2010 global sales (in millions) [according to Gartner]

Manufacturer	Sales	Market share
Nokia	100.3	33,15%
RIM	48.8	16,13%
Apple	47.5	15,70%
Samsung	23	7,60%
HTC	21.5	7,11%
Others	61.5	20,32%
Total	302.6	100%

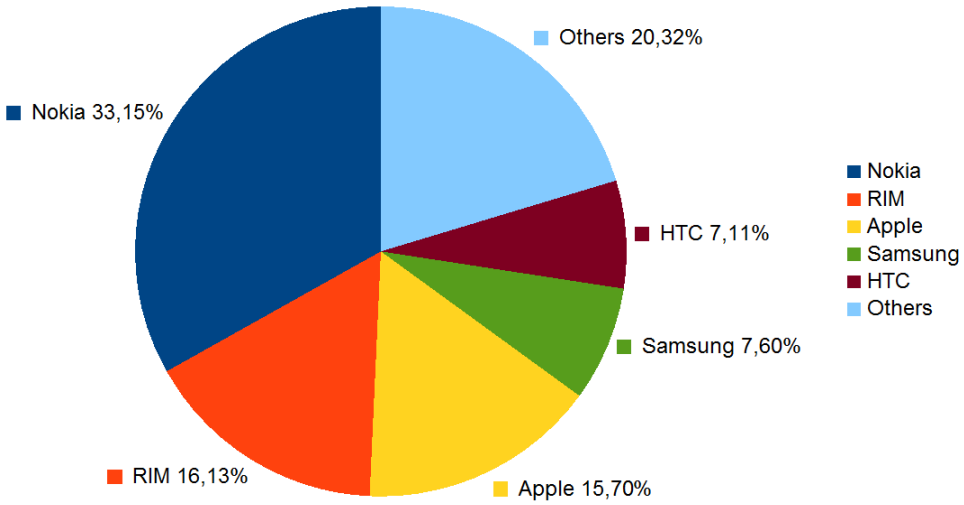

Figure 2.2: Top five smartphone manufacturers by 2010 global sales [according to Gartner]

People pay the largest price for devices from Apple, HTC and RIM. At the begin-

ning of this year the average mobile phone price was 105€. Nevertheless, the growth of smartphones' popularity depends on decrease of their price [3]. Yet, in the market the main players are those who can offer the cheapest multi-functional phones to consumers.

2.1.3 Operating systems

To properly manage hardware and software resources of smartphone an efficient operating system (OS) is needed. Manufacturers choose operating systems based on their demands from one of actually existing systems such as *Android*, *Symbian* or *Windows Phone*, or have designed and developed a whole new platform for their phones like Apple did with *iOS*, and *RIM* with *BlackBerry OS*. This paragraph will depict main features of five most popular smartphone operating systems, which are: Android, BlackBerry OS, iOS, Symbian and Windows Phone.

Android

Android is a comprehensive open-source platform for mobile devices, mainly phones and tablets. It offers complete software stack that includes an operating system, which separates hardware from the programs that run on it, hence it is possible for many devices to run the same applications. Additionally, the user experience can be improved with customization of the system.

A developer designing applications needs only to have tools and framework provided with Android Software Development Kit (SDK), no physical phone to test programs is necessary (unless some specific sensors which are hard to emulate are required). Android gives to the developer a powerful Application Programming Interface (API) for managing User Interface (UI) and hardware capabilities of the device [4]. A designer uses Java programming language in his work.

Android is becoming the world's most popular mobile platform. It lets user fast browsing, syncing with cloud, multi-tasking and gives the possibility of downloading and sharing thousands of applications stored on *Android Market*[5].

BlackBerry OS

BlackBerry OS is an embedded operating system dedicated for BlackBerry mobile phones developed by RIM. It is a multitasking environment designed to work with track wheel, track ball, track pad, QWERTY keyboard and touch screen inputs.

Smartphones with this system are designed to function mainly as personal digital assistants, Internet browsers, media players and devices for entertainment purposes. They are well known for the ability of receiving and sending encrypted e-mails through secure channels. RIM provides to the user the BlackBerry Messenger which is a high security level program for instant messaging.

Tools for developing applications and themes for BlackBerry are available for everyone on developer's website [6]. A programmer uses mainly Java programming language but it is also possible to write applications in HTML5.

Programs can be downloaded on smartphones with BlackBerry OS via *BlackBerry App World* - RIM's application store.

iOS

The iOS is a Unix-like operating system designed for Apple mobile devices, originally developed for iPhone smartphone, but in the end embedded also to iPod touch, iPad and Apple TV. It is licensed only for installation on Apple hardware.

Multi-touch user interface is designed to be intuitive, reliable and attractive. It is based on direct manipulation with gestures and an immediate response to them. Another feature of iPhone devices is the efficient battery with an effective energy management. Apple provides to the user a possibility to use iCloud (cloud storage), Siri (intelligent software assistant) and iPod media player functionalities as an added value.

The iPhone app developers are provided with iOS SDK which includes Xcode development environment and iPhone simulator. The SDK can be installed only on Apple Mac OS-based computer [7]. All applications are written in Objective-C with a possibility to program some elements in C or C++.

The *App Store* is Apple's digital distribution mean for iOS applications. It contains more than 500,000 programs which have been downloaded more than 18 billion times. The iPhone is a very famous but also a very expensive smartphone, hence it is not the most popular platform.

Symbian

Symbian is a computing platform for mobile devices designed uniquely for smartphones [8]. Currently it is the most popular operating system for cellulars. Symbian Trademark belongs to the Symbian Foundation Ltd. - in the past it was non-profit organization founded by Nokia, Sony Ericsson, NTT DoCoMo, Motorola, Texas Instruments, Vodafone, LG Electronics, Samsung Electronics, STMicroelectronics and AT&T to steward the Symbian platform development. Nowadays Symbian OS is maintained by Accenture on behalf of Nokia.

The platform satisfies consumers demands and offers multiple home screens, gesture interaction and multitasking. It also allows the user fast web browsing, recording and playing multimedia and editing .doc, .xls and .pdf files.

A developer can create rich applications in one of two frameworks provided - Qt SDK and Symbian Platform SDK [9]. Created programs can be run on simulator or on a physical device. C++ programming language is used.

Nowadays mainly Nokia devices run the newest Symbian OS versions, hence applications for Symbian are downloaded mostly via *Nokia Store*.

Windows Phone

The Windows Phone is another very powerful system for mobile platforms [10]. It is the *Windows Mobile* successor with redesigned fundamentals of functioning. Since it is rather a new operating system, and Microsoft requirements and constraints for devices from manufacturers are well defined, it has not become extremely popular yet, but everything is about to change in the following years.

Capabilities of Windows Phone devices are very rich. Everything on the phone is grouped in so called hubs (containers of functionalities of the same type). The use of many applications at one time is also possible (multitasking). Microsoft provides mobile version of selected *MS Office* programs as an added value to the user. System is designed on Metro UI paradigm which is based on clear and intuitive interaction with the consumer, hence achieving great user experience.

Microsoft prepared very good programming environment for developers. Just by installing Windows Phone SDK one can get everything needed for application writing, testing and releasing. Windows Phone SDK consists of Microsoft Visual Studio 2010 Express for Windows Phone, Windows Phone Emulator, SDK assemblies and libraries and Microsoft Expression Blend SDK for Windows Phone for designing layout. The .NET C# and .NET Visual Basic are used as programming languages.

Marketplace is Microsoft's application store, where programs can be published, sold and downloaded from. Each day new programs are released on Marketplace which means that platform gains popularity.

Smartphone OSs in years 2009-2015

Industry of cellular phones changes rapidly. Throughout the years Symbian has been the most popular platform for mobile devices chosen by such concerns as Sony Ericsson, LG, Motorola, Samsung and Nokia. Symbian has been loosing the market share over the years, especially after the release of the first Android-based platform in 2008 and expansion of rival operating systems.

Table 2.3 shows past, present and forecast market share of OS for smartphones. Nokia's Symbian is no longer number one OS for smartphones. Android has become the most popular platform for mobile devices at the end of last year.

Last year sales in USA show that Google's Android platform was most frequently chosen by consumers. The second and the third Americans' favorite device came from Research In Motion (BlackBerry OS) and Apple (iOS) respectively. European market trends were as follows - still the main popular devices came from Nokia with Symbian on board, but Android-based terminals has been becoming gradually more popular.

Table 2.3: Worldwide smartphone operating system market share in 2009 - 2015 [according to Gartner]

OS	2009	2010	2011	2015
Android	3,90%	22,70%	38,50%	48,80%
BlackBerry OS	19,90%	16,00%	13,40%	11,10%
iOS	14,40%	15,70%	19,40%	17,20%
Symbian	46,90%	37,60%	19,20%	0,10%
Windows Phone	8,70%	4,20%	5,60%	19,50%
Others	6,10%	3,80%	3,90%	3,30%
Total devices sold (in millions)	172	297	468	631

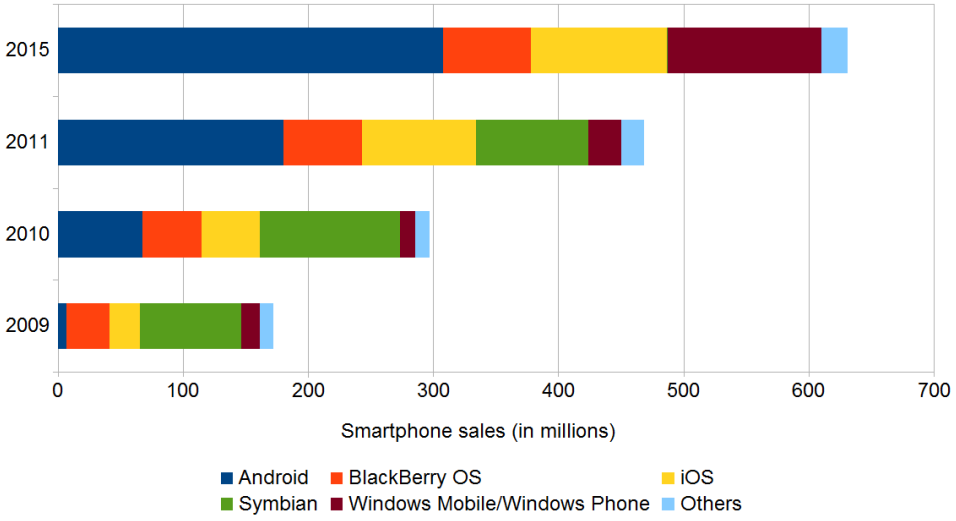

Figure 2.3: Smartphone sales in years 2009-2015 [according to Gartner]

According to Gartner analysis depicted in the figure 2.3, it is displayed that Android will remain as a leading operating system for smartphones.

2.1.4 Capabilities

Mobile handsets have become very powerful devices to manage consumer's demands. Today people do not use their phones just to make phone calls or to send text messages. One can do plenty of activities with their intelligent cellular. Available functions are: easy web-browsing, fast e-mail sending and receiving, capturing high quality photos and videos, chatting or making video conferences, listening to the music collection, entertaining with many games, using business applications and location based services. Almost all devices are equipped with the following functionalities:

- GSM/GPRS/EDGE system compatibility,

- HSPA/WCDMA system compatibility,
- Bluetooth connectivity,
- Wi-Fi: IEEE 802.11b/g/n connectivity,
- GPS signal receiver (with A-GPS),
- multi-task operating system,
- multi-touch event responsiveness,
- video camera,
- accelerometer,
- gyroscope,
- digital compass.

Smartphone is a device which handles many wireless interfaces. A user can be connected to GSM/UMTS network to use telecommunications services or use Wi-Fi connectivity for Internet actions. Another functionality is the use of GPS(Global Positioning System)/A-GPS(Assisted-GPS) signal receiver which permits to obtain one's position quickly. All devices run multi-task operating systems and interact to multi-touch events of the user. Standard equipment for handsets are also video camera, accelerometer, gyroscope and digital compass.

2.1.5 Future

As mentioned before, smartphone future market penetration depends mainly on decreasing its price. Achieving small price is possible with Android platform which can be operated on many devices with no constraints about their specifications.

Smartphones are becoming very computing-efficient devices. Manufacturers has begun to install multi-core processors in new models of their devices. Most of new terminals are also becoming able to work in LTE networks.

Energy consumption of smartphones has always been an issue. Battery life between two adjacent charging times could be even less that 24 hours with high utilization level. Accumulators capacity has improved over the years, but this is still not enough. Manufacturers came up with many energy-saving concepts, where some of those are: Active-Matrix Organic Light-Emitting Diode (AMOLED) technology, solar batteries, converting mechanic into electric energy from user's interaction with the phone.

Applications exploiting concept of LBS (Location-Based Service) and augmented reality are becoming very relevant in user's life. Almost every smartphone has an embedded GPS chipset, few of them are also GLONASS (GLObalnaya NAvigatsionnaya Sputnikovaya Sistema) enabled. With the release of European Galileo satellite navigation system, the higher accuracy, reliability and precision can be achieved by devices equipped with Galileo signal receiver. According to European Union prognosis, by the end of year 2020, 90% of mobile phones will be Global Navigation Satellite System (GNSS) enabled.

Figure 2.4: Nokia HumanForm concept

Figure 2.4 shows one of Nokia's future phone concept by Nokia Research Center. Since design will also play key role in phone industry, the concept is smooth and delicate. Vision is that handset responds to bend and touch events. The *HumanForm* depicts flexibility of forward-looking devices.

2.2 eCall system (by Michał Wojtysiak)

European Union currently holds 3 main projects for integrated emergency assistance. A voice only instance, that is already implemented and fully operational is a *112* number [11]. Rather than calling a particular type of service like police or ambulance, an emergency is reported and the proper services are distributed by *112* regional call center operator [12].

More recent *E112* is enhanced by the position obtained from cellular base stations. Additionally to a voice call being established, coordinates with limited accuracy are provided to the operator [13].

The most complex and efficient is the latest *eCall* system, targeted to automotive sector exclusively (figure 2.5). The *eCall* combines *E112* voice call with additional Minimum Set of Data (MSD) message transmission. Such message contains information about the position and car parameters [14].

System functioning relies on mounting dedicated devices in cars with an independent GNSS receiver, a cellular network transceiver and with a sensor unit. An alarm is sent automatically when a severe car accident was detected by sensors (e.g. airbag launch) or manually by a driver. The unit responsible for processing received

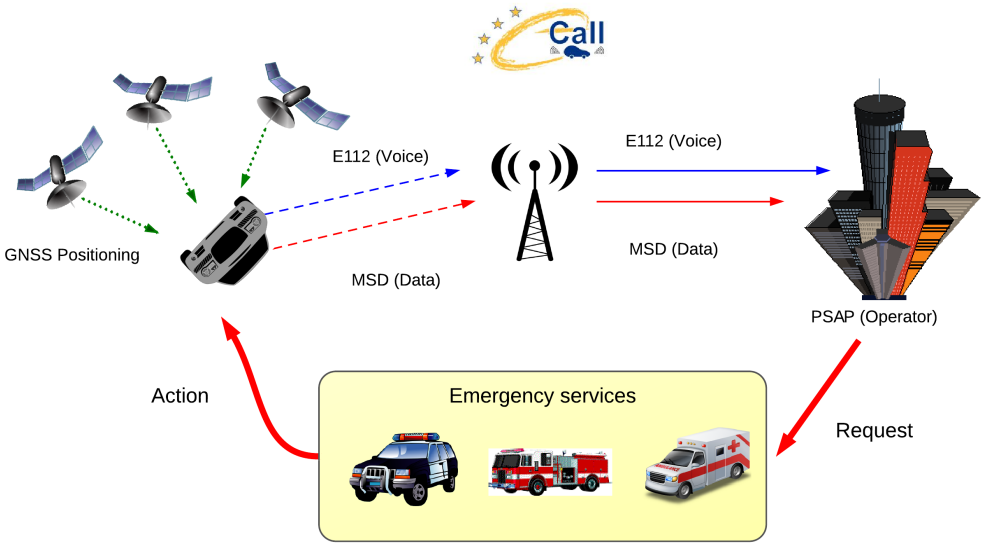

Figure 2.5: eCall system scheme

connection and data as well as sending emergency services is Public Safety Answering Point (PSAP) [15].

The main requirements for eCall modems are speed and reliability. When MSD is transmitted, the voice connection is muted. As for emergency situation, the duration of muting should be minimized. However, it is also essential that MSD is transmitted in reliable way, i.e using Cyclic Redundancy Check (CRC) scheme. Details about transmission requirements and tests of eCall proposal modem made by 3GPP can be found in [16].

2.3 ICE applications

Smartphones without applications are just devices to call, text, receive e-mail or browse Internet. Environment changes with programs on board - a telephone becomes a platform for entertainment, work, business and protection. Programs installed on the phone create an effective ecosystem where the main entity is the terminal itself.

In Case of Emergency (ICE) is the term corresponding to the concept of enabling paramedics, SAR or other public services to contact mobile phone owner's relatives in case something bad happens to him. It can be done by tagging appropriate contacts in phone book with "ICE" label or with a special application, where a user can add ICE contacts and store his medical information.

Figure 2.6 lists some of ICE applications available on Android Market [5]. They share similar functionalities but are designed with different approaches, also the layout differs. On the Windows Phone Marketplace and App Store not many positions were

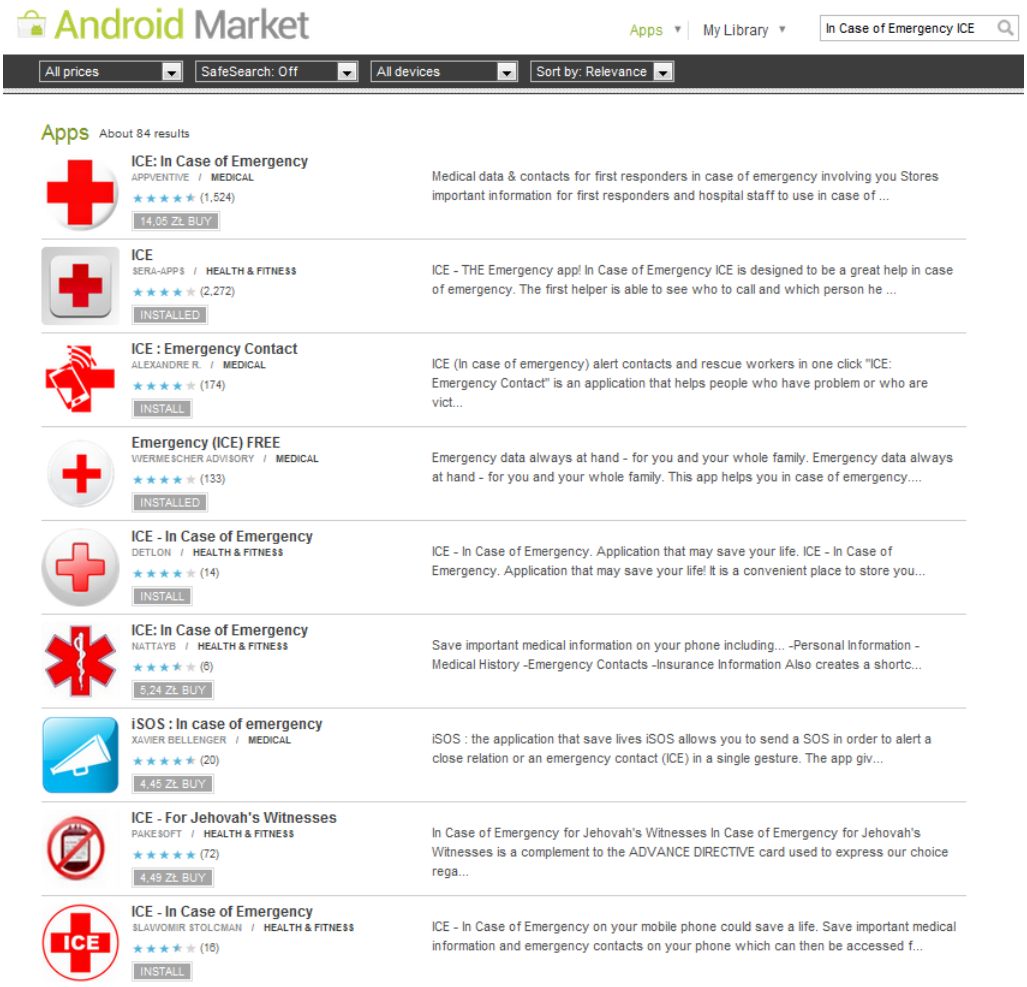

Figure 2.6: List of ICE applications on Android Market

noticed. In this work the Emergency (ICE) application by Wermescher Advisory will be described as the representative of ICE programs on Android Market.

Emergency (ICE) application by Wermescher Advisory is a personal emergency data storage. Apart from standard functionalities such as list of people to call and health information database it offers:

- emergency data for the whole family,
- personal details and photo of the person,
- special instructions,
- insurance information,
- information about one's doctors,
- emergency numbers of different countries (manual choose),
- reminder to update emergency data (after set time of inactivity),
- "Emergency" widget for lock screen (available with external application).

Another applications which can be crucial in case of danger are those allowing a user to call embassy or emergency numbers. On Android Market only one application with all world's embassy database exists, and there are some with specific country embassy list.

The *TravelSafe Pro* is an application for Android, which contains a list of country emergency and embassies numbers. Access to crucial information is rather fast, but a user has to do everything manually. It is also necessary to set the nationality in order to access embassy info. When a user taps the back button, he or she has to navigate backwards through settings pages, an activity which can be quite vexing.

Chapter 3

Life-saving application

3.1 Introduction

Android is the best platform for mobile application development. The main goal of this work is to write an implementation of a program, which puts into use chosen eCall functionalities for an Android-based smartphone. Its creation process is described in this chapter. In the following life-saving application features, the scheme, the programmer's environment along with the Android application fundamentals will be revealed. Further sections of this chapter discover in details the application itself along with layout design.

3.2 Main concepts

3.2.1 Life-saving application features

The main function of written application is to be a rescue interface between the user and SAR services. The eCall essential feature is the accident detection and providing geo-location information. Car crash is discovered by the sensors, but the alarm can be also set up manually by pressing the *panic* button. After that, an emergency call is established.

According to this scenario (without an accident detector and sensors), some functionalities of eCall system can be implemented in a life-saving application. The key point is the user's position information. Geographic coordinates are the universal language that everyone can speak and understand, hence they should be visible for the user. An alarm can be switched on manually by tapping the SOS button. After that, the public service number can be dialed and one can inform about danger status and his or her position.

A person which has got injured or has become unconscious, is often unable to communicate with rescue services. If paramedics arrive to help such an individual, they have no information about him. An application for a smartphone is the place where medical, insurance and personal information can be stored, in order to help medical services quickly get to know how to provide help to that specific person.

Another useful feature is the implementation of an internal phonebook of contacts to be called in case of emergency, i.e. when relatives should be notified about the status of smartphone owner.

ICE applications already exist on the market. They provide the user's medical database, ICE contacts phonebook and allow calling emergency numbers, but nothing more than these. The goal of a life-saving application is to be an intelligent, multi-functional program. Simple ICE applications force the user to manually set or choose emergency numbers. Knowing one's location, it is possible to get from the database those emergency numbers, corresponding to country he or she stays in. All without the necessary interaction.

Since a target application should be multi-functional, it will also consist of a database of embassies numbers. It is a crucial function for people who are often in travel.

To stress out the features once again, all functionalities required in life-saving application are the following:

- user's latitude and longitude visible on the screen,
- panic button to call public services,
- medical, personal and insurance information storage,
- ICE contacts phonebook,
- database of embassies numbers,
- intelligent choosing of emergency numbers,
- all databases held on user's device (once installed there is no need for Internet connection to get data).

The design of the application should be simple and attracting to provide the great user experience. The application deserves an appealing name as well. The author will sustain the naming convention after *YETI snow* project mentioned in *Chapter 1*, hence this application is named **YETI life**, which is an acronym for *Your Emergency and Tracking Interface*.

3.2.2 Application scheme

A developer shall design his application with taking into consideration screens visible for the user, features mentioned in previous section, and the interactions between them. Figure 3.1 presents the scheme of all application functionalities divided into components, navigation and dependencies between them.

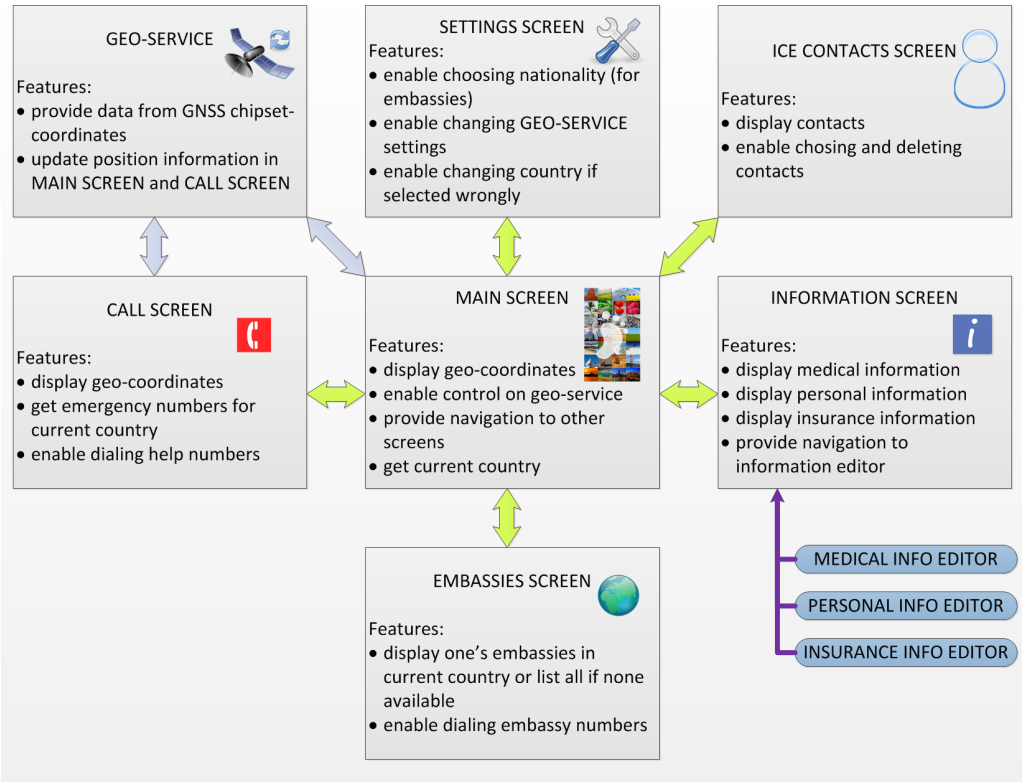

Figure 3.1: YETI life scheme

3.3 Programmer's environment

3.3.1 Java

Java is a general-purpose programming language which syntax derives from C/C++ languages. It has class-based architecture with simple object model. Developer's written code is compiled to Java byte code which runs on Java Virtual Machine (JVM). Since all Java applications work on virtual machine they can be run on any device with Java environment installed, no matter what the operating system is. Thanks to simplicity and portability that Java gives, it is one of the most popular programming languages. It is also the language used in Android programming.

This paragraph describes three concepts of Java used in life-saving application, which author considers worth-mentioning: CamelCase naming convention, names of class member objects and static variables. Java language concepts and information can be found in [17].

CamelCase is a naming convention mainly for class names. Multi-word class names should follow the pattern, where each word in the name begins with capital letter: "*EachWordBeginsWithCapitalLetter*".

Names for class member objects follow the convention where each name is preceded by the lower-case "*m*" letter. So every variable name considered as class variable guides "*mMember*" scheme.

Normally when a number of the same class objects are created, each instance owns a copy of class variables. A class variable tagged with *static* keyword is attached to a class as a whole, not to the object itself. It means that if a static class variable is changed for an instance, it is changed also for other instances of the same object class. The author uses static variables in life-saving application to update and to exchange information between different Activities, which is explained in further sections of this chapter.

3.3.2 Android application fundamentals

Android

Android is an open source system based on Linux. Thanks to that it gains hardware abstraction layers and can be imported to many devices. Since the beginning, the system has been improved and now offers new features with each version release. Figure 3.2 and table 3.1 depict how many percent of Android devices are equipped with particular OS version. More than a half of them work under control of Android 2.3.x (Gingerbread).

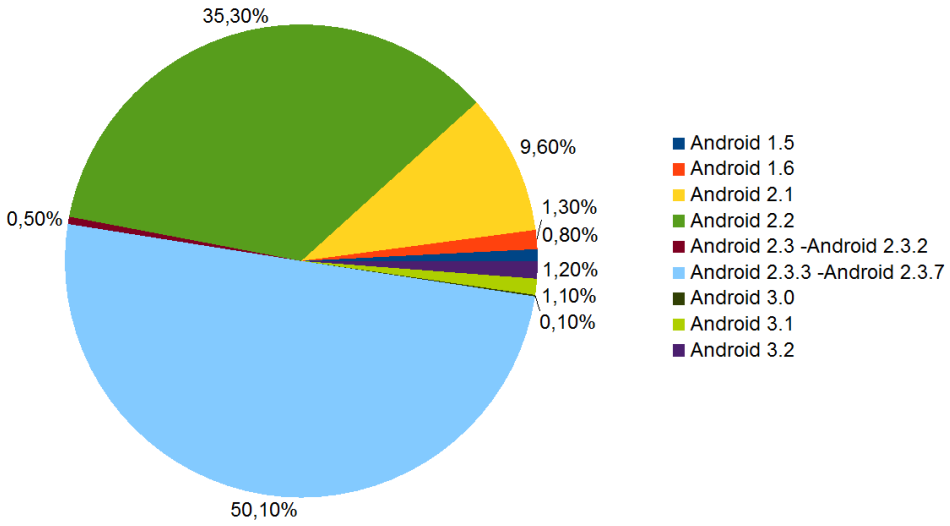

Figure 3.2: Android version distribution with data collected by Google during a 14-day period ending on December 1, 2011

The *API Level* is a value that uniquely identifies the API framework version corresponding to Android OS version. The life-saving application targets *API Level* 7 which corresponds to Android 2.1 Eclair. The author has decided to write a program

Table 3.1: Android version distribution with data collected by Google during a 14-day period ending on December 1, 2011

Platform	Codename	API Level	Distribution
Android 1.5	Cupcake	3	0,80%
Android 1.6	Donut	4	1,30%
Android 2.1	Eclair	7	9,60%
Android 2.2	Froyo	8	35,30%
Android 2.3 - Android 2.3.2	Gingerbread	9	0,50%
Android 2.3.3 - Android 2.3.7	Gingerbread	10	50,10%
Android 3.0	Honeycomb	11	0,10%
Android 3.1	Honeycomb	12	1,10%
Android 3.2	Honeycomb	13	1,20%

on Eclair-based device in order to target more consumers. Each newer Android system is backward compatible, which means it can run applications written for older system versions.

For the developer, the SDK offers Android Java classes. A programmer has access to most of Java's components and to an extension to specific user interface libraries designed for Android only.

Compiled Android program is an .APK file. It is a package which usually consists of such components as:

- Dalvik executable
- Resources
- Android Manifest

The three items will be discussed in detail later in this chapter. Dalvik executable file contains application code to be executed on Virtual Machine (VM). Resources are all media exploited in project, e.g. pictures and sounds. The Android Manifest is the declaration of all components used in project. The .APK package can optionally contain native code written in C/C++.

Dalvik Virtual Machine

Dalvik is a virtual machine constructed exclusively for Android. The device can run many Dalvik VMs efficiently at the same time. Files written by a programmer are compiled to Java byte code and then recompiled to Dalvik byte code. The virtual machine executes .dex files (Dalvik Executable). Each application works on its own entity of the Dalvik VM, which runs in its own process.

Activity

Single application's screen seen by the user can be referred to as an *Activity*. An activity consists of a window with its own user interface to interact with the

consumers. Every application is composed of a main activity - the entry point of the program. A developer can design a program with many activities and navigate between them. Activity life cycle is managed by Activity Manager responsible for launching, destroying and taking care of transitions between different states of the activity, allocating memory for UI components and drawing layouts.

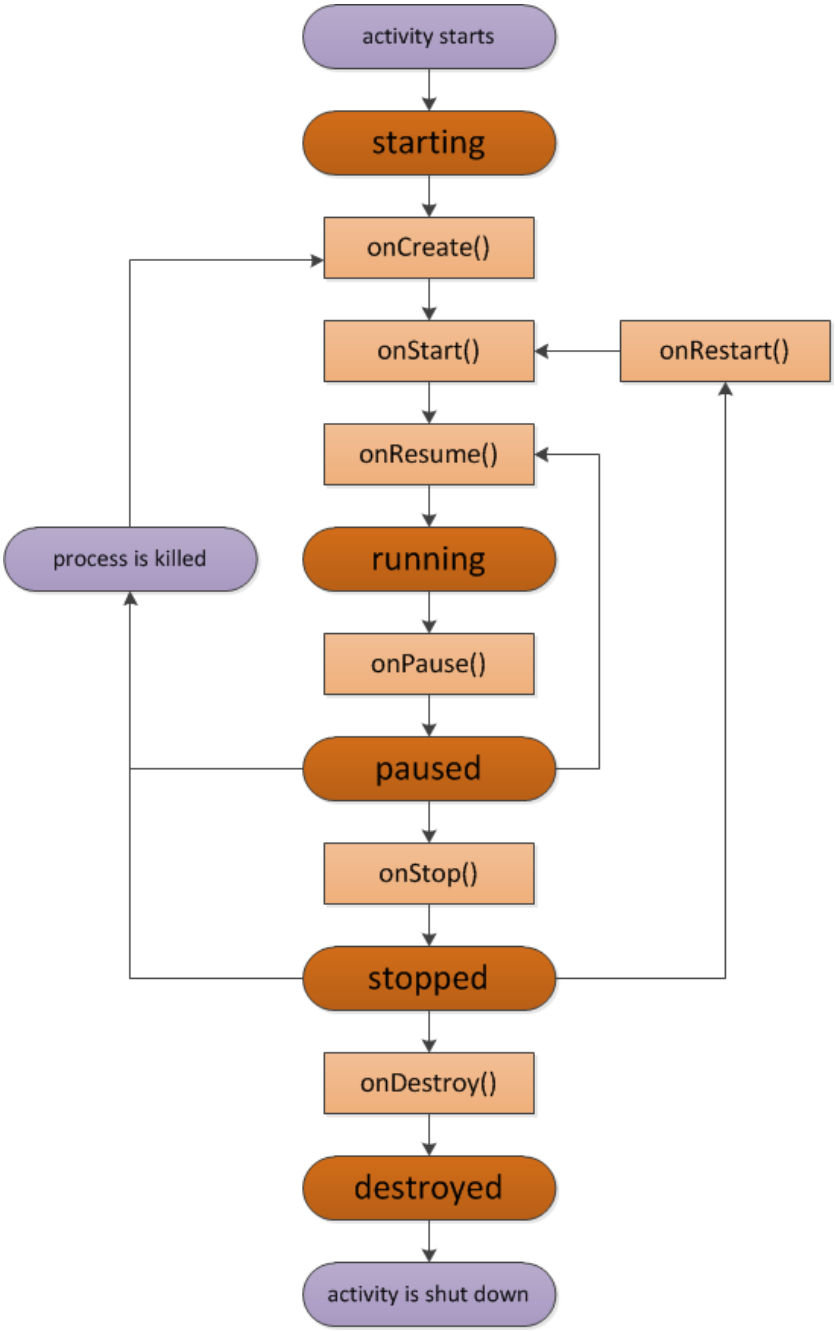

Figure 3.3: Activity life cycle

An activity can be in one of five states:

- starting - activity is not in memory and has to be brought into life,
- running - activity is visible and ready for interaction with a user,
- paused - another activity is in the foreground but paused activity is still partially visible and completely alive,
- stopped - activity was brought to a background but still exists in memory, another activity was launched on top of it,
- destroyed - activity no longer exists in memory.

Figure 3.3 shows the entire activity life cycle along with callback methods between state transitions. When activity starts the *onCreate()*, *onStart()* and *onResume()* methods are called. The activity is in the *running* state. If another activity goes in foreground, the *onPause()* method is executed and the previous state of activity is *paused*. If the activity becomes invisible to the user, the *onStop()* method is called and the activity becomes *stopped*. If the callback chain was executed to shut down an application, the *onDestroy()* function is performed and the activity is *destroyed*. From the *paused* state, the activity can come back to the foreground, hence *onResume()* method is called. The system can run out of memory. If so, a process can be killed. If the user navigates back to activities recently found in *paused* state, the *onCreate()*, *onStart()* and *onResume()* chain is executed once again. If *stopped* activity goes in the foreground after the process termination, the *onRestart()*, *onStart()* and *onResume()* chain is called.

The developer can easily manage life cycle of activities. To do so, some of the callback methods have to be implemented. Author of this thesis in his application implements mainly *onCreate()* and *onDestroy()* methods which is enough to sustain flexibility of transitions within application. All activities are listed and detailed in *Section 3.4*.

Service

A Service is an application component for background operations. Since service is invisible for the user it does not have a UI. Started service continues to work as long as the task is not finished or as long as it is not stopped explicitly. It runs also if the invoking component is destroyed or if a user navigates to another application. The service can perform same actions as an activity.

Another application component such as an activity can bind to the service to interact with it. Nevertheless, the author does not use such an approach in his application, hence bound form of service will not be described. Information about service binding can be found in [18].

Figure 3.4 depicts life cycle of a service. A service is started when it is invoked by an activity with *startService()* method. It has to be done explicitly using an intent, what will be described further in this work. The *onCreate()* and *onStartCommand()*

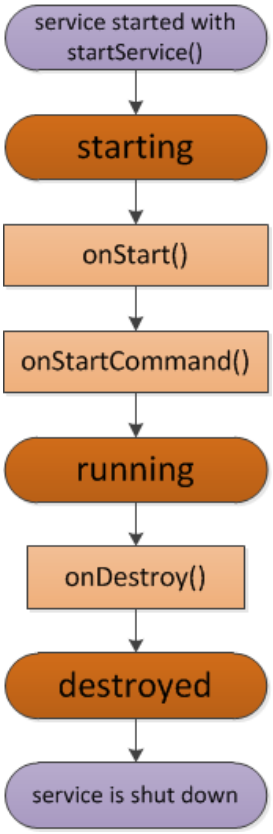

Figure 3.4: Service life cycle

methods are called. After they are performed, a service is in the *running* state. It is destroyed when it finishes its work or when it is ended by an activity. When *onDestroy()* method is executed, a service is shut down.

Life-saving application consists also of a location service. Description of the program and the service themselves will be detailed later. The author in his service implements all three callback methods mentioned.

Intents

Intent is a message exchanged between major components of an Android application. It contains the description of the operation to be executed. This structure is used also as trigger to start an activity or to start and stop a service. It is possible to include in an intent some specific data to be passed, such as the results of performed actions.

An intent can be explicit or implicit. In life-saving application only explicit intents are used. The following code excerpt shows how to create an intent object and how to explicitly start an activity.

```
Intent i = new Intent(YETIlifeMain.this, Call.class);
```

```
startActivity(i);
```

In the previous example the *YETIlifeMain* activity starts the *Call* activity. To do so, in *YETIlifeMain* class an intent object has to be initialized. The variable *i* of type *Intent* is brought to life using *Intent()* constructor which takes as arguments the sender class and the recipient class. Then the variable *i* is passed as an argument to *startActivity()* function which invokes the desired activity.

3.3.3 Eclipse

Eclipse is an open source Integrated Development Environment (IDE) dedicated for many programming languages. It contains also the collection of programming tools for Java. It is recommended for Android developers since an Android Development Tools (ADT) plugin for the Eclipse IDE exists. The plugin extends Eclipse capabilities and boosts Android application development. Author has used Eclipse IDE for life-saving application development.

Project structure

The developer's work is organized by Eclipse in projects. Those are stored in a workspace in a specific location indicated. Project name is usually written with CamelCase conception.

Figure 3.5 shows life-saving application project structure. Project consists of:

- *src* folder - contains a package with all created by the programmer Java classes,
- *gen* folder - contains a package with auto-generated R.java class file (described later),
- *Android x.x* - Android libraries (x.x corresponds to Android target version),
- *assets* folder - an additional folder for application content (not used by the author),
- *bin* folder - a container for compiled code,
- *res* folder - a container for resource items (described later),
- *AndroidManifest.xml* - declaration of application building blocks (described later),
- *proguard.cfg* - an automatically generated file for ProGuard tool used to optimize program code,
- *project.properties* file.

Packages are Java mechanisms for organizing class files of program. They provide visibility diversity between class structures in different packages. The package name should be the inverse of one's domain. If author's domain would be *application.example.com* the package name would get *com.example.application* name. The *src* folder contains *com.yetisolutions.yetilife* package with all classes created to make application work as assumed. Its content will be discussed later in this chapter.

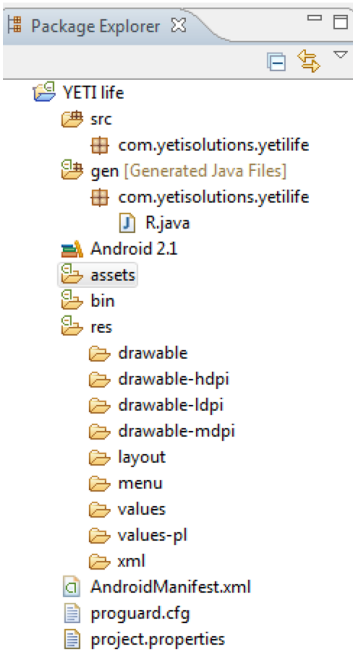

Figure 3.5: Eclipse life-saving application project

Resources and R.java class

The project's *res* folder is a container where all non-code resources of an application should be put. It consists of such folders as:

- drawable,
- drawable-*x*dpi (where *x* substitutes *l*, *m*, *h* letters),
- layout,
- menu,
- values,
- values-*xx* (where *xx* corresponds to ISO 639 two-letter language code),
- xml.

In the project all graphic resources are put in one of the *drawable* folders. Four *drawable* directories exist. First of them, the "pure" *drawable*, corresponds to all non-image resources, e.g. .xml files with shapes and color descriptions. In *YETI life* project this folder contains the description of tab layout for different selection states (used in *InfoTabHolder* tab activity). Android devices can be divided to handsets with high/medium/low-density screens. Drawables supporting those densities go to corresponding drawable-*h*/*m*/*l*dpi folders. In *YETI life* only *drawable-hdpi* directory is used. It contains all graphic files used for assuring great user experience: images of flags of countries, logos, buttons and icons.

The activity layout and the menu are described in special .xml files contained by corresponding to them folders. String values used in application are stored in */values/strings.xml* file. Layout, menu, values and xml folders will be explained deeply in *Section 3.3.4*. Layout of application's activities will be explained in details in *Section 3.4*.

The R.java file is an auto-generated class which can be related to as a lookup table containing resource names translated into unique integer ID for each of them. This ID can be used to reference a resource object from a program code. It is forbidden to modify R file manually. It is recreated every time one adds a new item to *res* directory.

3.3.4 XML

Extensible Markup Language

Extensible Markup Language (XML) is a universal presentation language based on, as its name says, markups. The data is represented in structures created with XML elements. XML element consists at least of "<"bracket, element name and ">"bracket. Optionally, such a tag can contain attributes of an element. Each XML document has to contain the XML declaration and exactly one main element, referred to as root element, holding other child items. XML is the universal language since one can define markups' names and attributes to describe data structures.

XML has been adopted also in many APIs as an optimal language for definition of non-program-code components. In Android it is used to describe application's main building blocks, layouts, menus' look, string and dimension values. More about Extensible Markup Language can be found in [19].

Android Manifest

Android Manifest is a special file where all activities, services, permissions and other components information shall be declared. Figure 3.6 shows fragment of YETI life Manifest (Android Manifest.xml) file. It consists of several XML tags and their attributes. First *<?xml (...) ?>* tag is a standard XML declaration. It contains XML version and encoding information. The *<manifest (...)>* tag is a container for all items necessary to declare for an application. Its attributes specify Android namespace (always *xmlns:android="http://schemas.android.com/apk/res/android"*) and describe main package name, internal version code and version name number visible for the users. The developer has also to declare minimum SDK version necessary to run his app. In *<uses-sdk (...)/>* it is set to *7* (*Eclair* as a target). The following three *<uses-permission (...)/>* labels are the required permission specifications. YETI life needs three allowances to:

- obtain user's position (*android.permission.ACCESS_FINE_LOCATION*)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.yetisolutions.yetilife"
          android:versionCode="1"
          android:versionName="1.0">

    <uses-sdk android:minSdkVersion="7"/>

    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.CALL_PHONE"/>
    <uses-permission android:name="android.permission.READ_CONTACTS"/>

    <supports-screens android:anyDensity="false"
                      android:largeScreens="true"
                      android:resizeable="false"
                      android:smallScreens="false"
                      android:normalScreens="true"/>

    <application android:icon="@drawable/icon"
                 android:label="@string/app_name"
                 android:theme="@android:style/Theme.Light.NoTitleBar">

        <activity   android:label="@string/app_name"
                    android:name=".YETIlifeMain"
                    android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>

            (...)

        <service    android:name=".GPSService"
                    android:exported="false">
        </service>

    </application>
</manifest>
```

Figure 3.6: Excerpt from YETI life Manifest

- call telephone (i.e. emergency) numbers (*android.permission.CALL_PHONE*)
- get access to user's phone book (*android.permission.READ_CONTACTS*)

Information about support of different screens is also specified in *<supports-screens (...)/>* tag's attributes of the manifest. The *<application (...)>* container attributes define reference to application's icon and label string hold in */res* directory. The *android:theme* attribute points out the style of displaying UI. YETI life application layouts are drawn taking whole available screen with no title bar (i.e. no label) displayed. Activities are declared with *<activity (...)>* tag. This label consists of activity name beginning with a "." mark (*.YETIlifeMain* in this example), supported screen orientation of the device (only portrait mode is supported by this application)

and a label to be displayed on title bar. All other application activities should be specified in the same way. Optionally, <*intent-filter*> can be specified in <*activity (...)*> container. It is a filter to decide whether called activity should respond to that invocation or not. Inside an <*intent-filter*> the intent action and category has to be specified. The *.YETIlifeMain* activity is the entry point to the application. It is invoked only if the application is launched (*android.intent.category.LAUNCHER*) and is used as top-level entrance to a program (*android.intent.action.MAIN*). YETI life Manifest declares also a location service in a <*service (...)*> tag. It is named *.GPSservice* and cannot be started by external applications (*android:exported="false"* attribute).

Layout

All layout files are stored in */res/layout* directory. Each of them is an XML document describing the look of the activity or other program components.

```
<LinearLayout
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_marginLeft="@dimen/logo_margin"
    android:layout_marginRight="@dimen/logo_margin"
    android:layout_marginTop="@dimen/logo_margin"
    android:layout_marginBottom="@dimen/gap5">
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="top"
            android:layout_marginRight="@dimen/logo_margin"
            android:src="@drawable/bigicon"/>
        <TextView
            android:id="@+id/textViewTitle"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="top"
            android:gravity="top"
            android:textColor="#000000"
            android:textSize="@dimen/title_font"
            android:text="@string/app_name"/>
</LinearLayout>
```

Figure 3.7: Fragment of YETIlifeMain activity layout file

Every visible UI component in Android is arranged in different layouts and views. Figure 3.7 is an excerpt from YETIlifeMain activity layout description. The layout elements are grouped into *LinearLayout*, which arranges them in a single row (*android:orientation="horizontal"* attribute). The *android:layout_width="fill_parent"* and *android:layout_height="wrap_content"* attributes point out that the layout should take all the horizontal space available, but it shall wrap its content in the vertical direction. The layout's margins are specified with *an-*

droid:layout_marginX="@dimen/logo_margin" attribute, where *X* refers to *Top, Right, Left, Bottom* side and the attribute value is the *logo_margin* object stored in dimensions.xml file. The *ImageView* is a container for graphic resources. The source of an image to be drawn is specified in *android:src="@drawable/bigicon"* attribute. Layout elements can also contain *android:layout_gravity* and *android:gravity* properties. First one specifies how to place the element with respect to the parent, second one refers to arranging its own children inside the layout. *TextView* is a view displaying text. To refer inside program code to a layout or view object its id has to be specified (*android:id="@+id/textViewTitle"*). The text color, size and string to be displayed inside the view are indicated as well. All layout components' descriptions follow the previous pattern. More about Android layouts can be found in [18].

Menus

Menus are graphical UI components visible for the user when he or she taps the *MENU* button on the handset. Their description is similar to other layout's. Figure 3.8 shows *main_menu.xml* file content, which refers to YETIlifeMain activity menu. Menu contains items. Each of them has three attributes set: an id, a title and an icon to be displayed.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/settings"
          android:title="@string/settings"
          android:icon="@drawable/setting"/>

    <item android:id="@+id/exit"
          android:title="@string/exit"
          android:icon="@drawable/exit"/>
</menu>
```

Figure 3.8: YETIlifeMain activity menu description

Values

The */values* directory is a place where all dimensions and strings used in application can be stored. They are kept in *dimens.xml* and *strings.xml* files respectively. Both files have as a root *<resources>* element. The data is placed inside the *<dimen></dimen>* and *<string></string>* elements correspondingly. Application can support many languages. It is done via creation of a redundant *strings.xml* file stored in */values-xx* folder, where "*xx*"is a two-letter language code. YETI life supports English language (as default) and Polish language, hence Polish strings are stored in */values-pl/strings.xml*. The redundant file is a copy of the default strings.xml file, but the content inside *<string>* elements is translated to Polish. The file for Polish strings does not have to contain all of them, if so they are taken from the default file by the Android OS. The figure 3.9 shows examples of language support XML

files written for YETI life application. Appropriate strings values are chosen based on language settings of the device.

/values/strings.xml

```
<resources>
   <string name="app_name">yeti life</string>
   <string name="country_temp">tap to set country</string>
   <string name="call">call</string>
   (...)
</resources>
```

/values-pl/strings.xml

```
<resources>
    <string name="country_temp">dotknij żeby ustawić kraj</string>
    <string name="call">zadzwoń</string>
    (...)
</resources>
```

Figure 3.9: The example of English and Polish strings.xml file

XML as database holder

XML is a great structure for holding database items, thanks to its flexibility and versatility. In the detailed description of YETI life application the exploited XML databases of countries, emergency numbers and embassies are revealed. This data files are stored in */xml* directory.

3.4 YETI life application

YETI life is an implementation of life-saving application depicted on the scheme in the figure 3.1. It comprises the following components (.java files stored in *com.yetisolutions.yetilife* package):

- YETIlifeMain activity (MAIN SCREEN in the figure 3.1),
- Call activity (CALL SCREEN in the figure 3.1),
- AllEmbassies, Embassies and EmbassDetails activities (to provide EMBASSIES SCREEN functionalities in the figure 3.1),
- InfoTabHolder tab activity (INFORMATION SCREEN in the figure 3.1) with different tabs: MedTab, PerTab, InsTab activities,
- MedicalEdit activity (MEDICAL INFORMATION EDITOR in the figure 3.1),
- PersonalEdit activity (PERSONAL INFORMATION EDITOR in the figure 3.1),
- InsuranceEdit activity (INSURANCE INFORMATION EDITOR in the figure 3.1),
- Contacts activity (ICE CONTACTS SCREEN in the figure 3.1),
- MySettings activity (SETTINGS SCREEN in the figure 3.1),

- CountrySelector activity for nationality and country selection,
- GPSservice service (GEO-SERVICE in the figure 3.1),

Additionally, YETI life application contains the following databases (.xml files stored in */res/xml* directory):

- database of countries (described in *Section 3.4.1*),
- database of emergency numbers (described in *Section 3.4.1*),
- database of embassies (described in *Section 3.4.1*).

and data types:

- Country class (to represent country as its name and ISO code),
- Nationality class (to represent nationality as its name and ISO code),
- Embassy class (to store embassy info),
- ICEcontact class (to store name and number of ICE contact),
- MedInfo class (to store medical info),
- PerInfo class (to store personal info),
- InsInfo class (to store insurance info),

3.4.1 Databases

The databases of YETI life are stored in */res/xml* directory. The reference from the program code to an .xml file is shown by the following snippet (obtaining countries.xml database):

```
getResources().getXml(R.xml.countries);
```

Database of countries

This database is a container used to get user's country based on MCC (Mobile Country Code) knowledge (procedure described in YETIlifeMain section). The MCC is a unique country identifier in which the wireless telephony network works. Root element of XML database of countries is simply *<country>*. The database records are *<co>* elements and each of them consists of the following attributes: country name, two-letter ISO 3166-1 alpha-2 country code and MCC. Names for elements and attributes are shortened to contain as little space as possible. The following fragment of database of countries shows a record for the Afghanistan country.

```
<co name="Afghanistan" iso2="AF" mcc="412" />
```

To obtain desired data from the database, the file has to be parsed. The following code excerpt is used to create list of countries from an XML file.

```
XmlResourceParser xpp = getResources().getXml(R.xml.countries);
xpp.next();
int eventType = xpp.getEventType();
while (eventType != XmlPullParser.END_DOCUMENT)
{
 if (eventType == XmlPullParser.START_TAG)
 {
  String elemName = xpp.getName();
  if (elemName.equals("co"))
  {
   Country myCountry=new Country();
   myCountry.Name=xpp.getAttributeValue(null, "name");
   myCountry.ISO2=xpp.getAttributeValue(null, "iso2");
   myCountry.MCC =xpp.getAttributeIntValue(2,0);
   myCountries.add(myCountry);
   Countries.add(myCountry.Name+", "+myCountry.ISO2);
   if(mMcc== myCountry.MCC)
   {
    mIso2=myCountry.ISO2;
    mCountry=myCountry.Name;
   }
  }
 }
eventType=xpp.next();
}
```

XmlResourceParser is a java object behaving like an XML parsing interface. Document parsing is based on an events chain, where an event is the beginning or the end tag of an element. In the loop, as long as there is non-parsed data in the document, xpp object searches for *<co>* element. If the element is found, the desired attributes are extracted from it. The data is written to *Countries* list of *Country* objects class. Additionally, if current MCC code is the element's MCC, the current country is set.

Database of emergency numbers

The database of emergency numbers is a structure for holding telephone numbers to the police, firefighters, medical services and, if applicable, other public services for all world's countries. To describe the building blocks the excerpt from the database of emergency numbers is shown below:

```
<emergency_phonebook>
   <continent name="europe">
     <country name="Albania" iso2="AL" mcc="276" main="127">
        <service name="police" no="129"/>
        <service name="medical" no="127"/>
        <service name="fire" no="128"/>
     </country>
     (...)
</emergency_phonebook>
```

The database is held by <*emergency_phonebook*> root element. All countries are grouped by their continents. The <*country (...)*> tag is the same as the <*co (...)*> tag in the database of countries, but with an additional attribute of main number included (e.g. 112). The <*country (...)*> element has <*service (...)*> child descendants. The <*service (...)*> tag is an emergency number record with its name and number specified. Obtaining the emergency numbers data follows the scheme presented in the previous paragraph.

Database of embassies

During the time of writing of this thesis, there are 247 countries and autonomy territories on Earth. To provide numbers of embassies for all of them, the 247 .xml files have to be constructed and maintained. Up to now only the database of Polish embassies was created. The structure of single entry in database of embassies is shown below.

```
<embassies country="Poland" iso2="PL" mcc="260">
 <embassy country="Afghanistan" iso2="AF" mcc="412" prefix="93">
  <city name="Kabul" description="Ambasada Rzeczypospolitej Polskiej
      w Islamskiej Republice Afganistanu" address="10 Maghzan Street,
       Kart-e Seh, KABUL, Afghanistan">
     <telephone number="796554372"/>
     <telephone number="798292170"/>
     <telephone number="797329"/>
  </city>
  (...)
</embassies>
```

The holding element <*embassies (...)*> has three attributes to check if appropriate file to parse was chosen (country name, ISO 3166-1 alpha-2 country code and MCC number). The <*embassies (...)*> keeps <*embassy (...)*> elements (with the same check-parameters to know whether appriopriate country was selected). Each <*embassy (...)*> can have <*city (...)*> children element in which city name, embassy description, address and telephone numbers are stored. Access to the data is similar as in the previous database described.

3.4.2 *Main* screen

In this section the *YETIlifeMain* activity will be precisely discussed. Other application components will not be described in detail since they adapt the same patterns. Only their particular cases will be revealed.

YETIlifeMain is the entry point to the program. It means that when YETI life application is launched, this activity is the first screen visible for the user. This activity offers navigation to other screens of main functionalities and control of Location service.

Layout

Application layout is based on Windows Phone Metro concept. It is designed to be modern, clean and intuitive. Figure 3.10 shows the main screen of YETI life application.

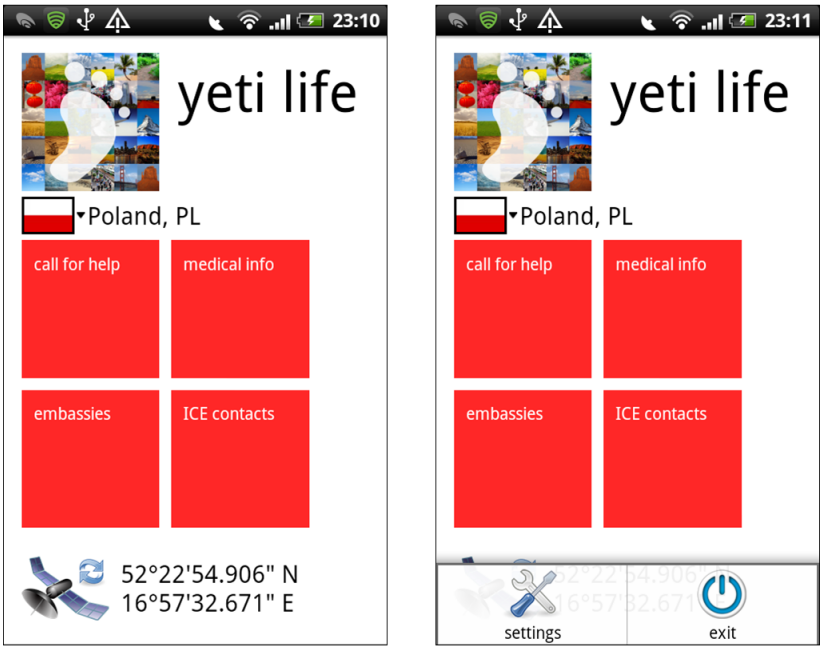

Figure 3.10: Main activity layout

In all application components the YETI life logo is displayed in top-left corner. From this window user can navigate to *Call*, *Embassies*, *Information* and *ICE contacts* screens by pressing corresponding button. If the country chosen by the application is not the one that user currently stays in, it can be changed by tapping the flag image. The user can also access special features when MENU button on device was pressed - one can navigate to settings or exit YETI life program.

Code explanation

YETIlifeMain is a java class which extends Activity class. All structures and objects used throughout the application are initialized in the overridden *onCreate()* method. The code snippet presented below contains Android program and *YETIlifeMain* activity key points to be explained.

```
public class YETIlifeMain extends Activity
{
    int     mMcc;
    public static ArrayList<Country> myCountries;
    public static ArrayList<String>  Countries;
```

```
    public static TextView            mTextPosition;
    public static Button              mGPSButton;
    Button    mTileCall,
              mTileEmbassy,
              mTileInfo,
              mTileContacts;
    public static final String PREFS_NAME = "YETIlifeSettings";
    public static SharedPreferences settings;
    public static SharedPreferences.Editor editor;
    (...)

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
         super.onCreate(savedInstanceState);
         setContentView(R.layout.main);

         mTextPosition = (TextView) findViewById(R.id.textPosition);
         mTileCall = (Button) findViewById(R.id.imageTileCall);
         (...)
         settings = getSharedPreferences(PREFS_NAME, 0);
         editor = settings.edit();
         (...)
         mTileCall.setOnClickListener(new View.OnClickListener()
         {
          public void onClick(View v)
          {
            Intent i = new Intent(YETIlifeMain.this, Call.class);
            startActivity(i);
          }
              });
         (...)
         try
         {
          TelephonyManager tel = (TelephonyManager) getSystemService(
              Context.TELEPHONY_SERVICE);
          String networkOperator = tel.getNetworkOperator();
          if (networkOperator != null)
          {
             mMcc= Integer.parseInt(networkOperator.substring(0, 3));
          }
         }
         (...)
}
private void setFlag()
{
  if(mIso2!="iso2")
  {
     Resources res = getResources();
     String mDrawableName = mIso2.toLowerCase();
     int resID;
```

```
        Drawable drawable ;
        try
        {
          resID = res.getIdentifier(mDrawableName, "drawable",
              getPackageName());
          drawable = res.getDrawable(resID);
         }
        (...)
        mFlag.setImageDrawable(drawable);
}
(...)
}//end of YETIlifeMain activity class
```

First lines of the code in the *YETIlifeMain* activity class are the standard declaration of variables and structures used further. To initialize them, the *onCreate()* method from superclass has to be overridden. Inside it, the method in the superclass has to be called (*super.OnCreate(savedInstanceState)*). The designed layout for the activity is set by *setContentView(R.layout.main)*. With the next two lines the reference to layout's components are set up (they have to be casted to their types - *TextView* and *Button*). Also an access to shared preferences and its editor is initialized. SharedPreferences class serves to save and get primitive data (booleans, floats, ints, longs, and strings) in key-value pairs. The storage is private and persistent, which means that can be accessed only by "owner"application and is accessible even if the application was killed. This storage is used to save information data and user's nationality. Another important thing to do is providing an interaction with the user. The *mTileCall.setOnClickListener()* is a method which sets an action to be performed if one presses *call for help* button. The *onClick()* method specifies that the Call activity shall be invoked with an intent. Intents were described in *Section 3.3.2*.

The most interesting action performed is obtaining knowledge about user's current country. It has to be done intelligently, without his interaction. The idea is to get the MCC number, if a phone has an active GSM/UMTS network connection. With *TelephonyManager* class the information about the telephony services on the device can be accessed. The *tel.getNetworkOperator()* method returns six-digit operator code, which consists of MCC and MNC (Mobile Network Code). The MNC uniquely identifies a mobile phone carrier. The MCC code is extracted from the string and stored as a class member (*mMcc*).

After comparing mMcc object with MCC codes stored in database of countries and discovering actual country, its flag should be displayed. Images in */res/drawable-hdpi* directory can be accessed by having a *Resources* instance (*getResources()*). Image ID has to be found by *res.getIdentifier()* method. Then a reference to an image can be made (*res.getDrawable()*), a drawable can be set as a source of an *ImageView* (*mFlag.setImageDrawable()*).

3.4.3 Location Service

This application component has no UI because it runs in the background. The service obtains geo-position by using Android Location API. At the beginning and at the end of its work, the information about the current status is displayed in Toast notification. It also updates static main activity components responsible for showing one's current position.

Code explanation

The code fragment presented below shows key-points of *GPSService* class. At the beginning of the code, the main class's components are initialized. The *mLocationManager* instance is a *LocationManager* object used to access the system location services. The *mLocationListener* serves as listener to position change events. The *mLastKnownLocation* is used to store last location obtained by Android Location API. To use Android Location service it is necessary to choose location provider (GPS or Network). The *getProvider()* method returns string according to available location providers. If GPS provider is enabled it is returned in the first place, otherwise Network provider (or none) is returned.

```
public class GPSService extends Service
{
    private static final int METERS= 20;
    private static final int ACCURACY=10;
    private LocationManager mLocationManager;
    private LocationListener mLocationListener;
    public static String mCurrentLocation;
    Location mLastKnownLocation;
    String myProvider;
    String getProvider()
    {
     if(mLocationManager.isProviderEnabled(LocationManager.
         GPS_PROVIDER))
     return LocationManager.GPS_PROVIDER;
     (...)
    }
    @Override
    public IBinder onBind(Intent intent)
    {
     return null;
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId)
    {
     return Service.START_NOT_STICKY;
    }
    @Override
    public void onCreate()
    {
```

```
    mLocationManager =  (LocationManager) this.getSystemService(
        Context.LOCATION_SERVICE);
    myProvider=getProvider();
    if (!myProvider.equals(PROVIDERS_DISABLED))
    {
     Toast.makeText(getApplicationContext(), getString(R.string.
         GPSstart), Toast.LENGTH_LONG).show();
     (...)
     lastKnownLocation_GPS = mLocationManager.getLastKnownLocation(
         LocationManager.GPS_PROVIDER);
     (...)
     mLocationListener = new LocationListener()
     {
      public void onLocationChanged(Location location)
      {
       if(isBetterLocation(location, mLastKnownLocation))
       {
        setLocation(location);
        updateLocation();
       }
       if(((location.hasAccuracy()) ? location.getAccuracy() : 2*
           ACCURACY) < ACCURACY) stopSelf();
      }
      (...)
     }
     mLocationManager.requestLocationUpdates(myProvider, 0, METERS,
         mLocationListener);
    }
    private void updateLocation()
    {
     try
     {
      YETIlifeMain.mTextPosition.setText(mCurrentLocation);
     }
     catch (Exception e){}
     (...)//same for Call activity
    }
    @Override
    public void onDestroy()
    {
     (...)
     mLocationManager.removeUpdates(mLocationListener);
     (...)
    }
    (...)
}
```

The *onBind()* and *onStartCommand()* are standard Service methods. The *onBind()* method returns null, because it is not allowed to bind to *GPSService* for any activity. The value returned by the *onStartCommand()* means that if the application is killed, the service will not be restarted until the application will make this explicitly.

At the beginning of *onCreate()* method all the *GPSService* class members are declared. If some location provider exists, the service starts its work and Toast notification is displayed. It is done with *Toast.show()* method, where the toast content was set before. The *Toast.makeText()* takes as an argument the context, the string to be displayed and the duration of showing the message. At the start the service gets also the last remembered location stored by the Android *LocationManager*.

The *mLocationListener* object is initialized with its event handler methods. If user's position is changed, the *onLocationChanged()* method is called with the new location as an argument. If the new location is better than the previous one (in terms of accuracy, etc.) it is prepared to be displayed by *setLocation()* method and shown with *updateLocation()* function. If a position is found with the desired accuracy of 10 meters the service stops its work. To get location changes the request to location updates has to be done with *mLocationManager.requestLocationUpdates()* function, which takes as arguments: the name of the provider, the minimum time between updates in milliseconds, the minimum distance interval for notifications in meters and the location listener.

The *updateLocation()* method updates layout components in *YETIlifeMain* and *Call* activities. It accesses static class members and sets their values to current location. When the *GPSService* stops its work, the *onDestroy()* method is called - its *LocationManager* instance has to remove updates requested at the beginning of its work. It is done with *mLocationManager.removeUpdates()* method.

3.4.4 *Call* screen

From this screen user is able to call emergency services. The activity gets appropriate phone numbers from the XML database. The user's position is displayed as well, to give him the possibility to pass it as a message to public services.

Layout

Figure 3.11 depicts *Call* activity layout. It consists of four buttons for calling public services and a place where one's geo-coordinates are displayed. When the user taps the calling button a confirmation dialog is displayed.

Code explanation

The code excerpt below shows interesting particularities of *Call.java* class. Strings declared at the beginning of the class are used to store emergency numbers taken from the database.

```
public class Call extends Activity
{
    String mMain, mPolice,mFire, mMedical;
    String noToCall;
```

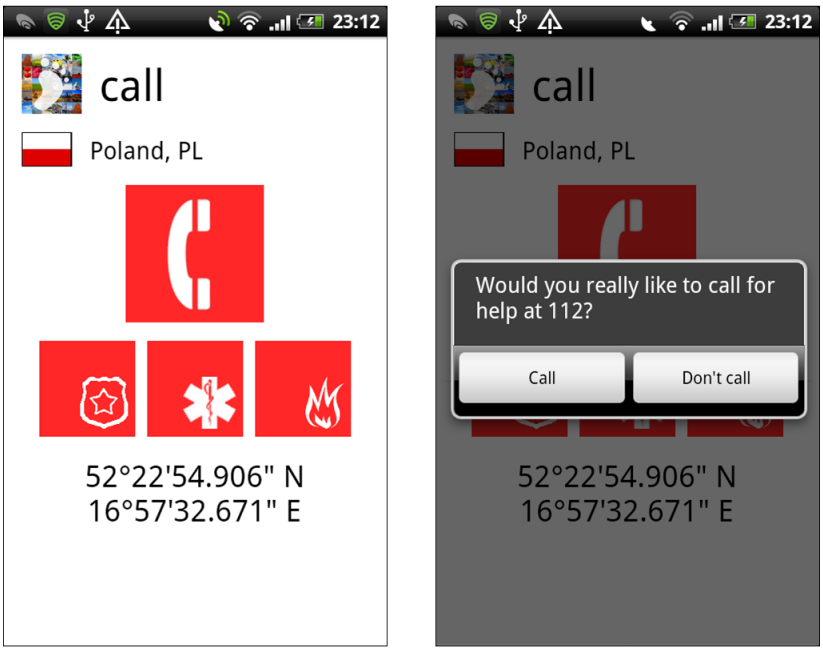

Figure 3.11: Call for help activity layout

```
private static final int DIALOG_CALL_ID = 0;

@Override
public void onCreate(Bundle savedInstanceState)
{
 (...)
 mPanicButton.setOnClickListener(new View.OnClickListener()
 {
  public void onClick(View v)
  {
   (...)
   noToCall=mMain;
   showDialog(DIALOG_CALL_ID);
   (...)
  }
 });
 (...)
 getEmergencyNumbers();
}
(...)
protected Dialog onCreateDialog(int id)
{
 AlertDialog dialog;
 switch(id)
 {
  case DIALOG_CALL_ID:
       String call=getString(R.string.callButton);
```

```
            String noCall=getString(R.string.dontCallButton);
            AlertDialog.Builder builder = new AlertDialog.Builder(this)
                ;
            builder.setMessage(getString(R.string.callDialog)+" "+
                noToCall+"?")
            .setCancelable(false)
            .setPositiveButton(call, new DialogInterface.
                OnClickListener()
             {
              public void onClick(DialogInterface dialog, int id)
              {
               try
               {
                startActivity(new Intent(Intent.ACTION_CALL, Uri.parse(
                    "tel:"+noToCall)));
               }
               (...)
               removeDialog(DIALOG_CALL_ID);
              }})
             .setNegativeButton(noCall, new DialogInterface.
                 OnClickListener()
             {
              public void onClick(DialogInterface dialog, int id)
              {
               dialog.cancel();
               removeDialog(DIALOG_CALL_ID);
              }
              });
             dialog = builder.create();break;
             default:
             dialog = null;
            }
            return dialog;
    }
}
```

When the user taps the button displayed in the middle of the screen, the main emergency number is set as number to call (*noToCall* variable) and the confirmation dialog is displayed (*showDialog()*). The *getEmergencyNumbers()* is a method which parses the external database of emergency numbers.

To show a confirmation dialog, the *onCreateDialog()* has to be implemented. As an argument it takes the *id* of the dialog to be shown. To build a dialog the *AlertDialog* and the *AlertDialog.Builder* objects are created. With the *AlertDialog.Builder* the dialog's message, buttons (and on click events) can be set. If user confirms a will of calling an emergency number the call action is performed. The dialog is removed after displaying it, because it has to be recreated with a different message when the other button is tapped. The *dialog* object is created and returned by *builder.create()* method.

3.4.5 *Information* screen

The Information screen is a class which extends TabActivity class. It actually is a component which comprises three other activities as its different tabs: *MedTab*, *PerTab* and *InsTab* classes. In this screen user's medical, personal and insurance information is displayed. The info can be edited while the *MENU* button is pressed and the The *edit* option is chosen. Data is stored in the shared preferences.

The *Medical information* tab is an activity for health data to be displayed. The user can share such information as: blood type, health issues, allergies, diabetes, directives, taken medicines, organ donor and other relevant info. To represent the data in the code the *MedInfo* type was created.

The *Personal information* tab is used to display user's own relevant data. One can print such information as: name and surname, birth date, eye color, height, ID number, passport number, personal number and particular signs info. To represent the data in the code the *PerInfo* type was created.

The *Insurance information* tab provides insurance data. The user is allowed to show his health, medical, social and travel insurance numbers. To represent the data in the code the *InsInfo* type was created.

Layout

The Information screen layout is shown in three tabs (each data type displayed in one tab). The data typed by the user is listed in *ListView* component. The user has the access to each data type by pressing the appropriate tab down the screen. The Information Editor inputs are listed in *ListView* as well. The layout is shown in the figure 3.12.

Code explanation

The code fragment shown below is taken from *InfoTabHolder.java* class. It depicts how the tabs can be attached to *TabHost*, that is a bar in which tabs are displayed.

```
public class InfoTabHolder extends TabActivity
{
    TabHost mTabHost;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
     mTabHost = (TabHost) findViewById(android.R.id.tabhost);
     String tab=getString(R.string.tab_medical);
     View tabview = createTabView(mTabHost.getContext(), tab);
     TabSpec spec;
```

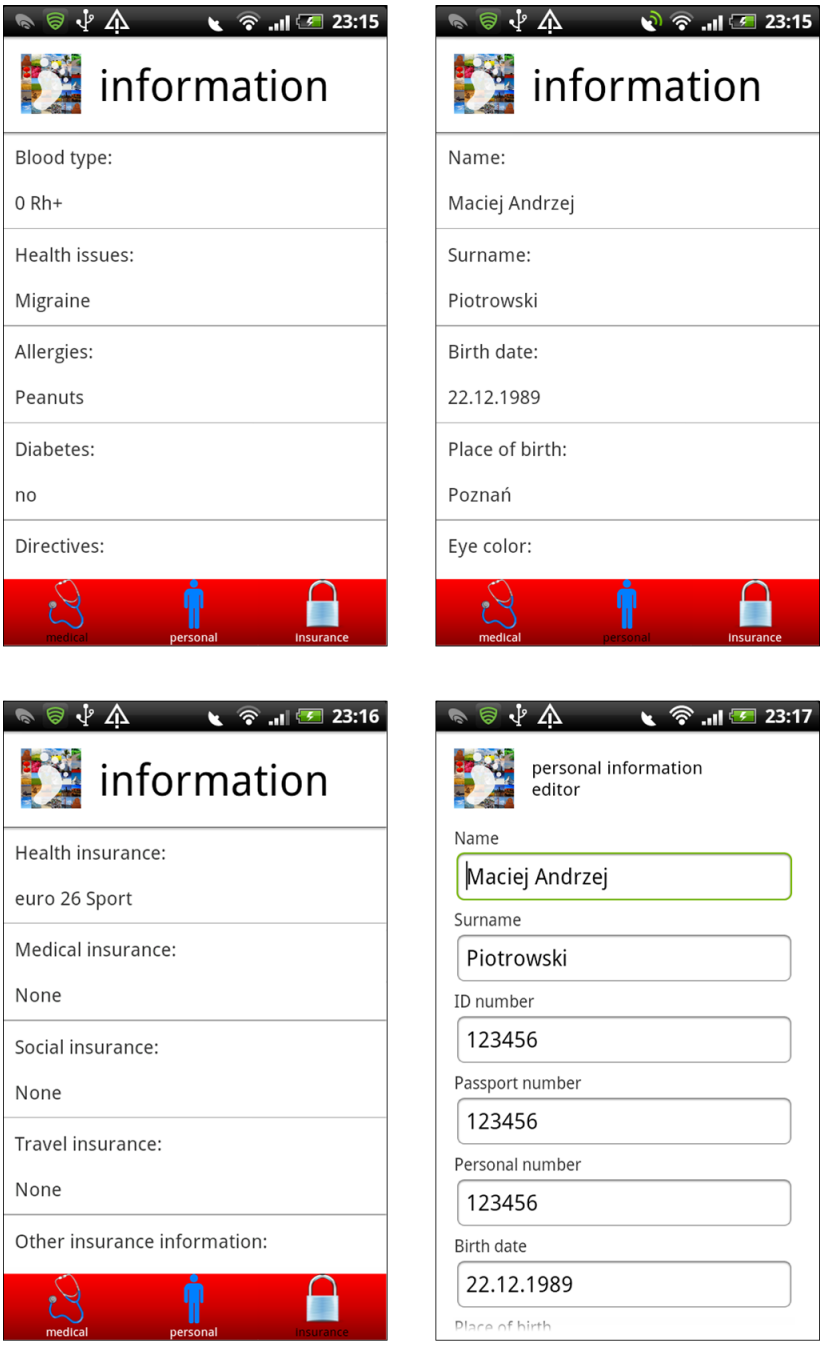

Figure 3.12: Information activity layout

```
 spec = mTabHost.newTabSpec(tab).setIndicator(tabview).setContent(
     new Intent().setClass(this, MedTab.class)).addTab(spec);
 (...)
 mTabHost.setCurrentTab(0);
}
```

```
    private View createTabView(final Context context, final String
        text)
    {
     View view = LayoutInflater.from(context).inflate(R.layout.
         info_tab, null);
     TextView tv = (TextView) view.findViewById(R.id.tabsText);
     tv.setText(text);
     ImageView iv= (ImageView) view.findViewById(R.id.TabIcon);
     if (text.equals(getString(R.string.tab_medical)))
     {
      iv.setImageDrawable(getResources().getDrawable(R.drawable.
          medical));
     }
     (...)
     return view;
    }
}
```

The *TabHost* is a container for a tabbed view. First of all the *TabView* has to be created with *createTabView()* method. This method takes as arguments a context to *TabHost* and a label string. Within the function a layout for a tab is inflated from resources. The label and the appropriate icon is set and the created view is returned. To compose the tab indicator, content, and tag (for tracking the tab) the *TabSpec* object is needed. All three tabs are initialized following the same pattern.

The code excerpts below explain how to access data in shared preferences. The *getString()* method from *settings* object in *YETIlifeMain* class is used. The method returns string object. It takes as arguments the name of the searched string (the *key*) and the default string value (returned if object not found in shared preferences).

```
mMyInfo.mBloodType=YETIlifeMain.settings.getString("blood", "not
    filled");
```

The example of putting data to shared preferences is presented below. The *putString()* method from editor object in *YETIlifeMain* class is used. It takes as arguments the *key* and its *value*. The changes have to be committed with *commit()* method from *editor* object in *YETIlifeMain* activity.

```
YETIlifeMain.editor.putString("blood",value);
YETIlifeMain.editor.commit();
```

The *edit* option is available from OptionsMenu. It can be created with *OnCreateOptionsMenu()*. To do so it just inflates layout from menu file (info_menu.xml) in resources. When a menu item is selected the *onOptionsItemSelected()* method is called. It handles all work to be executed for that menu option. In the options menu code excerpt below, an intent to *MedicalEdit* class is sent).

```
private static final int EDIT_REQUEST=3;
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
```

```
 MenuInflater inflater = getMenuInflater();
 inflater.inflate(R.menu.info_menu, menu);
 return true;
}
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
 switch (item.getItemId())
 {
  case R.id.itemEdit:
  Intent i = new Intent(MedTab.this, MedicalEdit.class);
  startActivityForResult(i,EDIT_REQUEST);
  return true;
  default: return super.onOptionsItemSelected(item);
 }
}
```

3.4.6 *Embassies* screen

The *Embassiess* screen was actually divided into three components: *AllEmbassies*, *Embassies* and *EmbassDetails* activities. The first one is responsible for listing all embassies in the database, the second is responsible for listing embassies available in the country the user currently stays in, and the third one displays detailed embassy information. If no database for user's nationality exists, the notification is displayed and the access is denied.

Layout

Figure 3.13 shows all Graphical User Interface (GUI) for embassy activities. For *AllEmbassies* and *Embassies* class it is a simple *ListView* structure. The window of *EmbassDetails* is composed of a call button and embassy information.

Code explanation

To store embassy information a special *Embassy* type was created. Embassies are stored in an *ArrayList* of type *Embassy*. To view embassies the *listEmbassies()* method is used. The XML database is parsed by *getEmbassies()* method. To show data on the *ListView* an *ArrayAdapter* of type *String* is required. The adapter is set with *mListView.setAdapter()* method, which takes *ArrayAdapter* object as an argument. Country names are displayed in the *ListView* as corresponding countries for embassies.

```
private void listEmbassies()
{
 myEmbassies=new ArrayList<Embassy>();
 Embassies=new ArrayList<String>();
 getEmbassies();
```

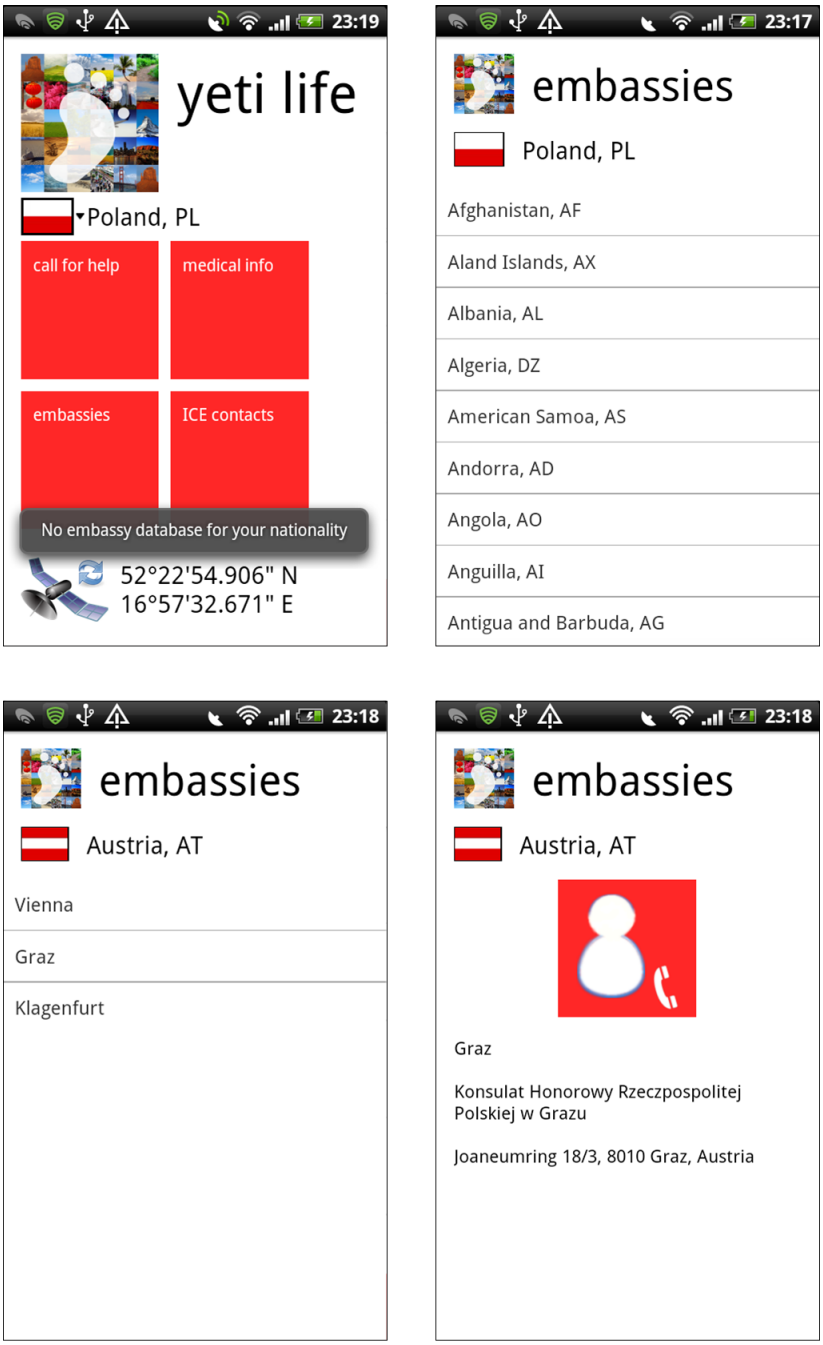

Figure 3.13: Embassies activity layout

```
mListView.setAdapter(new ArrayAdapter<String>(this, R.layout.
    country_item, Embassies));}
```

3.4.7 *ICE contacts* screen

The *Contacts.java* class represents an activity used to display numbers to be called when something bad happens to owner's handset. It provides a possibility of adding, removing and calling contacts from a list. When a contact is tapped, a dialog box with an action to choose is displayed (*call*, *delete*, *set as default*). All ICE contacts are stored in shared preferences. The *ICEcontact* type was created to store contact's name and number.

Layout

Activity layout is composed of a list showing all "in case of emergency" contacts and a button responsible for adding new entries in a phonebook. The activity look can be seen in the figure 3.14.

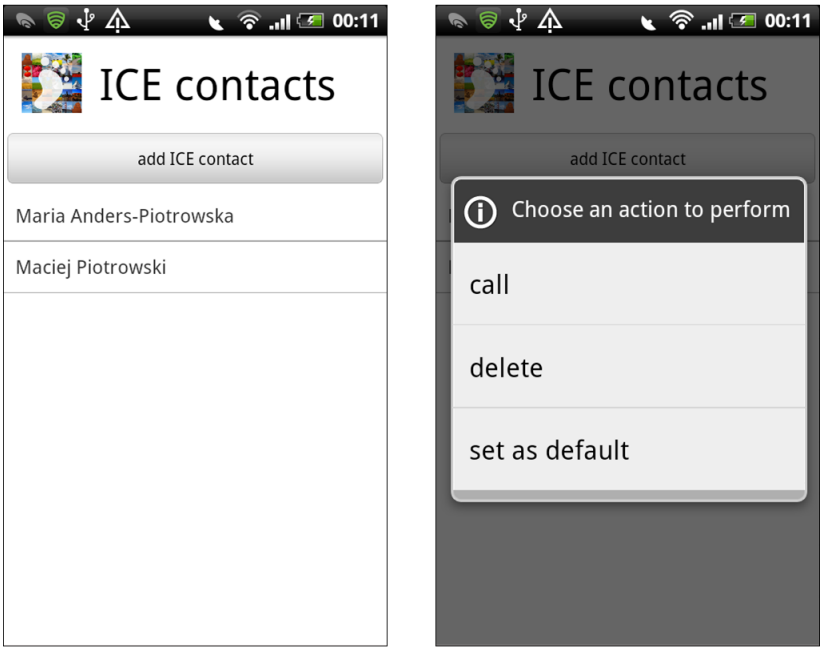

Figure 3.14: ICE contacts activity layout

Code explanation

To add a contact, application has to use *ContactsProvider*. The permission to access contacts is needed in *Android Manifest* file. An intent to phonebook is sent with *startActivityForResult()* method. When a record is chosen from the contacts, the *onActivityResult()* method handles picked data to be stored by application. The *onActivityResult()* method is explained in *Settings* activity description. To perform

a call from the application the same approach as in *Call.java* class was applied. The *ICE contacts* code excerpt is shown below.

```
void addContact()
{
 Intent contactPickerIntent = new Intent(Intent.ACTION_PICK, android.
     provider.ContactsContract.Contacts.CONTENT_URI);
 startActivityForResult(contactPickerIntent, CONTACT_PICKER_RESULT);
}
```

3.4.8 *Settings* screen

The *Settings.java* class is an activity where the user can change current country, nationality and Android Location Settings preferences. For storing country and nationality the special types were created (*Country.java* and *Nationality.java* classes respectively).

Layout

The Settings screen layout is displayed in the figure 3.15.

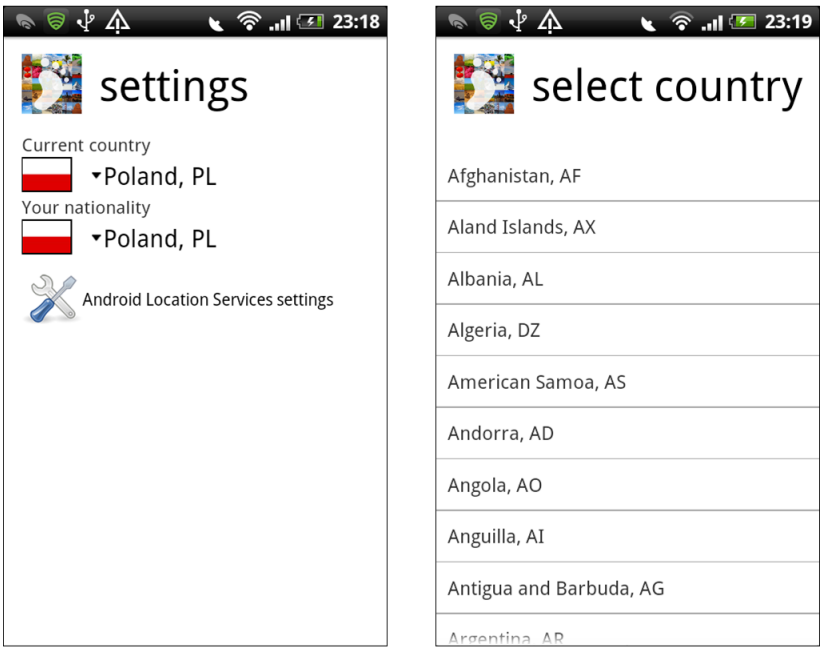

Figure 3.15: Settings activity layout

Code explanation

In *Settings* activity only two features are relevant and did not appear in previous descriptions. The *Settings* code fragment on the next page shows how to

change Android Location Settings. An activity specified by the Android internal *Settings.ACTION_LOCATION_SOURCE_SETTINGS* string is started.

```
public void onClick(View v)
{
 Intent myIntent = new Intent(Settings.
     ACTION_LOCATION_SOURCE_SETTINGS);
 startActivity(myIntent);
}
```

An activity result can be handled with *onActivityResult()* function. It takes as arguments the request code, specified when starting an activity for result, the result code and the intent. If *requestCode* equals the request code sent in *startActivityForResult()* method, and the activity *resultCode* is *RESULT_OK* the operations for that result can be made. The code below shows getting an activity result.

```
protected void onActivityResult(int requestCode, int resultCode,
    Intent i)
{
 (...)
 if (requestCode == SET_COUNTRY_REQUEST)
 {
  if (resultCode == RESULT_OK)
  {
   (...)
  }
 }
}
```

3.5 Summary

The fully-functioning YETI life application was created and described in this chapter. All assumptions were satisfied. The application was tested on HTC Desire HD smartphone working under control of Android 2.3.5 OS. The location service provides an accurate position within few minutes. In worst-case scenario it took 10 minutes, as measurement was taken in a moving bus in the city center. The application is reliable and handy.

Chapter 4

Wireless interfaces in Android

4.1 Introduction

Android is a very powerful OS which gives a plenty of opportunities to the application developer. The application programmer has a possibility to intervene in device's wireless interfaces: Wi-Fi and Bluetooth settings, get network information or obtain location data. In this chapter the access to phone components which use wireless technologies will be revealed.

4.2 Capabilities

Almost all mobile devices available on the market are equipped with Bluetooth card, Wi-Fi 802.11b/g/n and GPS chipsets. Also it is clear that all of the smartphones provide mobile telephone functionalities. In the Android OS one can interact with hardware components and access data provided by them.

4.2.1 GSM/UMTS network

First of all, a smartphone is a cellular network terminal which enables user to connect to GSM/UMTS network and benefit from its services (more about GSM and UMTS in [20]). A programmer has an access to information provided with the following object types:

- ConnectivityManager - to represent and to get network connection state,
- NetworkInfo - to get active network and its type, connection status and state information.

More about managing network connectivity can be found in [21].

4.2.2 Bluetooth

Bluetooth is a standard for wireless communications in short-range distances. It is mainly used for data exchange in Industrial, Scientific and Medical (ISM) band (from 2.4 to 2.48 GHz). More about Bluetooth can be found in [22].

Android programmer is allowed to: discover devices within range, manage peripherals connected to the terminal, control device's discoverability, communicate in peer-to-peer (P2P) mode over Bluetooth. To provide communications between devices they have to be paired. The following classes enable managing Bluetooth device:

- BluetoothAdapter - represents local device,
- BluetoothDevice - represents each external device,
- BluetoothSocket - used to create a connection between devices,
- BluetoothServerSocket - used as a listener for incoming connection requests.

The description of API components used for accessing Bluetooth devices and managing Bluetooth connectivity can be found in [21].

4.2.3 Wi-Fi

Smartphones have a possibility to create network connection using wireless card compatible with 802.11 b/g/n standard (more about IEEE 802.11 wireless standards in [23]).

Wi-Fi API in Android OS gives access to: detailed Wi-Fi and network information, monitor device's connectivity, intervene user preferences and make changes in Wi-Fi configurations, scan and list available Wi-Fi access points, provide Wi-Fi P2P communications (available in Android version 4.0 (Ice Cream Sandwich)). Programmer can interact with all components via the following object instances:

- WifiManager - used to represent Android Wi-Fi service, basic entity for managing Wi-Fi connectivity,
- WifiInfo - object containing: SSID, BSSID, MAC address, and IP address of access point the device is connected to, link speed and signal strength information,
- WifiConfiguration - object instance used to create or edit configurations and manage connections to networks.

The description of API components used for accessing Wi-Fi devices and managing Wi-Fi connectivity can be found in [21].

4.2.4 GPS

User can get location data provided by GPS signal receiver. GPS is composed of space segment (31 satellites surrounding Earth), control segment (control stations

on Earth) and user segment (e.g. GNSS-enabled phones). Satellites continually send messages containing: the time of message transmission, precise information about the orbit and so called almanac. More about GPS is described in [24].

One's current position can be found with GPS chipset or Google's cell-based location technology. The programmer can choose a technology to be used explicitly with its name or implicitly by defining accuracy, cost or other criteria.

To obtain one's position the basic elements have to be embedded in the code: *LocationManager*, *LocationProvider* and *LocationListener*. With basic instances of LBS objects the programmer can do the following:

- get current location,
- track device movement,
- set proximity alerts (detect motion in and out of selected area),
- check which location providers are enabled on the device.

Usage of main LBS components was described in section 3.4.3 of this work (Location Service in YETI life application).

4.3 SSID sensing

Service Set IDentifier (SSID) sensing is a concept created for the YETI ESNC 2011 project. It develops the idea, that alarm information can be spread in SSID sensed by other smartphones. Such an SSID contains SOS tag and one's position information. If it is detected by other devices, they set the same SSID to be sensed by the others. The figure 4.1 shows SSID sensing concept for mountain environment where the alarm data is passed to medical point in ski resort.

Unfortunately, even with Android OS on board setting a hotspot name (i.e. SSID) from program-code level is not possible at the moment. The concept with SSID sensing using Wi-Fi card has to wait until Android system developers will make this option available for application programmers. The user can turn on the hotspot option and set SSID manually with manufacturer's application. What can be done right now is the development of the same concept, but by using a Bluetooth technology. It is possible to maintain a Bluetooth connection for a range of even 50 meters (with good signal propagation conditions).

Bluetooth devices discoverability

The following code will extend life-saving application functionality to Bluetooth devices discoverability. The Device name will be set to SOS tag and position data (obtained with GPSService.java class mentioned in section 3.4.3). The concept with Bluetooth devices discoverability has not been tested yet. The code example was made based on *Bluetooth, Networks, and Wi-Fi* chapter in [21].

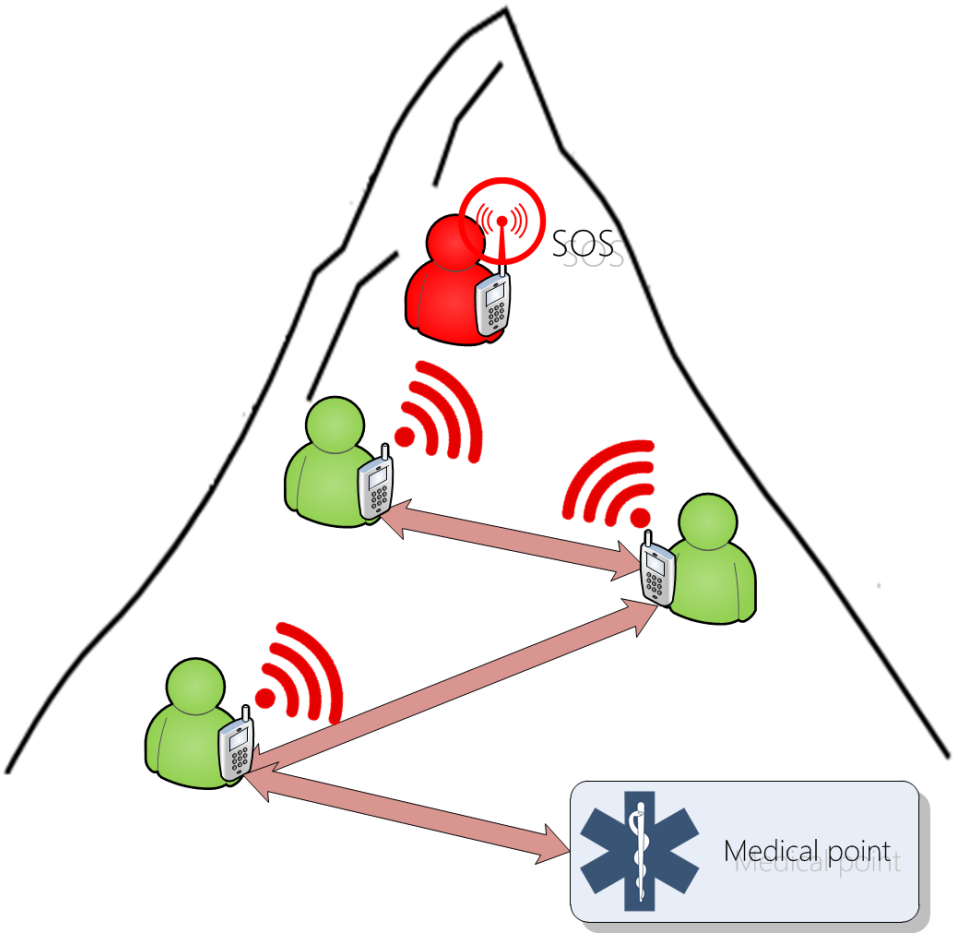

Figure 4.1: SSID sensing in mountain environment

The code excerpt below can be used to set the device name (corresponding SSID in project using Wi-Fi connectivity) and set the smartphone to be discoverable via Bluetooth (in 2 minutes time by default):

```
String myLocation;
(...)
BluetoothAdapter bluetooth = BluetoothAdapter.getDefaultAdapter();
bluetooth.setName("SOS"+myLocation);
startActivityForResult(new Intent(BluetoothAdapter.
    ACTION_REQUEST_DISCOVERABLE), DISCOVERY_REQUEST);
```

The *myLocation* string is a container for position information provided by *GPSService* service. To initialize *BluetoothAdapter* the *BluetoothAdapter.getDefaultAdapter()* method is used. Then one can set the device name (SOS tag and current location string in this case). The device can be discoverable after the intent with such a re-

quest is sent with *startActivityForResult()*. The following code is used for managing the behavior of device discovering peripherals in-range.

```
BroadcastReceiver discoveryResult = new BroadcastReceiver()
{
 @Override
 public void onReceive(Context context, Intent intent)
 {
  String remoteDeviceName = intent.getStringExtra(BluetoothDevice.
      EXTRA_NAME);
  (...)
 }
};
registerReceiver(discoveryResult, new IntentFilter(BluetoothDevice.
    ACTION_FOUND));
bluetooth.startDiscovery();
```

To get names of discovered devices the *BroadcastReceiver* has to be registered (*registerReceiver()* method). The *BroadcastReceiver* is an application component used to receive broadcast system messages. If a device is discovered an intent with its name is broadcasted. In override of *onReceive()* method this name is get with *intent.getStringExtra()*. The name can be parsed and set as own device name to spread the message further.

Chapter 5

Summary

YETI life is a unique life-saving program, which adapts selected eCall services to many environments such as city, village, forest, mountains, seaside and every place where cellular network connectivity can be provided. The main thesis assumptions have been fulfilled. The application for Android-based smartphone exploits wireless interfaces to call the rescue services (GSM/UMTS) and to obtain one's position (GPS).

The software gives to the user a possibility to alarm public services, inform about its current location and store medical information. As an extension one can also access database of embassies.

The reliability of solution depends on the connection with cellular network provider. If no carrier connectivity is available, no calls can be performed to notify the SAR services about one's danger.

LBS can provide accurate position in time up to ten minutes, depending on how far from current location the last coordinates where discovered, on the availability of last known location in phone's memory and on the usage of A-GPS technology or pure GPS signal. The accurate position can not be found in a building (GPS signal cannot be properly caught indoors). The user can turn off LBS if it is not needed. To preserve battery consumption the service stops itself if found position's accuracy is less than 10 meters. The accuracy information is available from GPS chipset.

SSID sensing concept described in *Chapter 4.3* cannot be implemented right now. The Android OS does not give to a programmer the possibility to set the SSID from the program-code level. The device can turn itself into a hotspot with set SSID only via manufacturer's application. Maybe setting SSID from program-code level will be possible in the near future with the release of new system versions. The scheme can be substituted with Bluetooth devices discoverability, which is only a theoretical concept.

The YETI life solution could be applicable worldwide with further development of

database of embassies, but their maintaining is problematic - there are many countries with many embassies around the world (databases for 247 countries with 247 entries each). Multilingual support has to be provided as well.

Bibliography

[1] M.Piotrowski and M.Wojtysiak, "European Satellite Navigation Competition website: YETI project description." http://www.galileo-masters.eu/index.php?anzeige=final11_bavaria.html, 2011.

[2] dotMobi, "Global mobile statistics 2011: all quality mobile marketing research, mobile Web stats, subscribers, ad revenue, usage, trends." http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats, 2011.

[3] D. Błaszczykiewicz (Leading Editor), "Mobile 2011," tech. rep., International Data Group Poland SA, 2011.

[4] Google, "Android Developers website." http://developer.android.com/index.html, 2012.

[5] Google, "Android Market website." http://market.android.com/, 2012.

[6] Research In Motion Limited, "BlackBerry - BlackBerry Developer Zone website." http://us.blackberry.com/developers/, 2012.

[7] Apple, "Apple Developer website." http://developer.apple.com/, 2012.

[8] Nokia, "symbian.nokia.com — Symbian at Nokia website." http://symbian.nokia.com/, 2012.

[9] Nokia, "Nokia Developer website." http://www.developer.nokia.com/, 2012.

[10] Microsoft, "Windows Phone website." http://www.microsoft.com/windowsphone/pl-pl/, 2012.

[11] "European Comission 112." http://ec.europa.eu/information_society/activities/112/ms/index_en.htm, 2011.

[12] "Factsheet 44: 112 - Your lifeline while travelling in the EU." `http://ec.europa.eu/information_society/newsroom/cf/itemdetail.cfm?item_id=5623`, 2011.

[13] "European Commission Factsheet eSafety." `http://ec.europa.eu/information_society/newsroom/cf/itemdetail.cfm?item_id=5623`, 2005.

[14] "4th eSafety Communication: eCall: Time for Deployment." `http://www.esafetysupport.org/en/ecall_toolbox/european_commission/index.html`, 2009.

[15] "Europe's Information Society, eCall infogramm." `http://ec.europa.eu/information_society/activities/esafety/images/ecall/adac_ecall_enl.jpg`, 2011.

[16] M. Werner and J. Huang, "Cellular In-Band Modem Solution for eCall Emergency Data Transmission," *IEEE*, 2009.

[17] E. Bruce, *Thinking in Java*. Prentice Hall, 4th ed., 2006.

[18] M. Gargenta, *Learning Android*. O'Reilly, 2011.

[19] E. Ray, *Learning XML. Guide to Creating Self-Describing Data*. O'Reilly, 2nd ed., 2003.

[20] K. Wesołowski, *Systemy radiokomunikacji ruchomej*. WKŁ, 2003.

[21] R. Meier, *Professional Android 2 Application Development*. Wiley Publishing, Inc., 2010.

[22] J. Bray and C.F. Sturman, *Bluetooth: Connect Without Cables*. Prentice Hall, 2000.

[23] B. O'Hara and A. Petrick, *The IEEE 802.11 Handbook: A Designer's Companion*. IEEE Standards Association, 2nd ed., 2005.

[24] E.D. Kaplan and C.J. Hegarty, *Understanding GPS: principles and applications*. Artech House, 2006.

YETI life installation

Preparing phone

Make sure that your smartphone runs Android OS version 2.1 or higher. To install application from .apk file one has to enable *Unknown sources* option in Android settings - it allows installation of non-Android Market programs. To do so, one has to navigate from the *Home screen* to *Menu* - *Settings* - *Applications* and enable *Unknown sources* check box.

Step-by-step installation of an application

1. Connect Android device to the computer with the USB cable (choose *Disk drive* connection type on your phone) or using Bluetooth technology.
2. Copy *YETI life.apk* file to your smartphone and remember the path to the file.
3. With file manager program access the *YETI life.apk* file on your device (author recommends *File Manager* application by Rhythm Software (`https://market.android.com/details?id=com.rhmsoft.fm` for accessing data).
4. Find and select *YETI life.apk*.
5. Tap *Install* button from newly opened activity
6. Wait for the *Application installed* notification and tap *Open* button to start the program.

To launch YETI life program select it from application list.

List of Figures

List of Abbreviations

Abbreviation	Description
EU	European Union
SAR	Search And Rescue
ESNC	European Satellite Navigation Competition
YETI (snow)	Your Entertainment & Tracking Interface
YETI (life)	Your Emergency Tracking Interface
PDA	Personal Digital Assistant
SIM	Subscriber Identity Module
RIM	Research In Motion
OS	Operating System
SDK	Software Development Kit
API	Application Programming Interface
UI	User Interface
LBS	Location Based Service
GPS	Global Positioning System
A-GPS	Assisted - GPS
AMOLED	Active-Matrix Organic Light-Emitting Diode
GLONASS	Globalnaja Nawigacionnaja Sputnikowaja Sistiema
GNSS	Global Navigation Satellite System
MSD	Minimum Set of Data
PSAP	Public Safety Answering Point
CRC	Cyclic Redundancy Check
ICE	In Case of Emergency
JVM	Java Virtual Machine
VM	Virtual Machine
IDE	Integrated Development Environment
ADT	Android Development Tools
XML	Extensible Markup Language
MCC	Mobile Country Code
MNC	Mobile Network Code
GUI	Graphical User Interface

Abbreviation	Description
ISM	Industrial, Scientific and Medical
P2P	Peer-to-Peer
SSID	Service Set IDentifier