Department of Electronic and Computer Engineering
Technical University of Crete

THESIS

"A new location-aware Android application for automatic synchronization of business and life calendars".

Liagouras Georgios Andreas

CHANIA

October, 2012

Advisor: Polychronis Koutsakis

ABSTRACT

Since the beginning of the new millennium cell phones have been quickly moving into our everyday life and becoming an important part of it. Cell phones have coherently changed the way we live and work. The significant technological advances led to the adoption of new and elaborated features to cell phones turning them into a mini computer, a mailing system, a text messenger, a video camera, even a game console.

Over the last couple years a major breakthrough happened, the development of mobile applications which was once a prerogative of cell phone manufacturers, has been made available to the world of developers for further improvements and innovation. This move marked the beginning of a new era for cell phones, the era of **smartphones**. Mobile applications with access to smartphones' hardware such as 3G/Wi-Fi, GPS, camera, offer services that have a great impact on the way we perform our daily tasks.

Motivated by this general concept, in this work we developed an application for the Android platform that reads from the user's calendar the locations, descriptions and scheduled times of his daily events and based on this data offers him the optimal route to visit these locations in the minimum time. For even better user-experience, algorithms that check estimated time of arrival (ETA) and current locations run in the background, in order to display pop up messages if the user has activated triggers while visiting a location. The app's target group is wide, from businessmen to ordinary people who face the complexity of everyday life.

ACKNOWLEDGMENTS

I would like to express my greatest gratitude to the people who have helped and supported me throughout my thesis. I am grateful to my supervisor Polychronis Koutsakis for his continuous support for the thesis, from initial advice and contacts in the early stages of conceptual inception and through ongoing advice and encouragement to this day.

I would also like to thank the members of my thesis committee, Prof Michael Lagoudakis and Prof Alexandros Potamianos, for their participation.

Special thanks to Amir Sayegh for his continuous support and co-supervision of this thesis through the exchange of ideas and alternatives on how to proceed with the design of the application.

Finally, I wish to thank my parents for their undivided support and interest who inspired me and encouraged me to go my own way, without whom I would be unable to complete my thesis. Last but not least but not the least I want to thank my friends who appreciated me for my work and motivated me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

SDK:Software Development Kit

AVD: Android Virtual Device

API: application program interface

TSP: Travelling Salesman Problem

NDK: Native Development Kit

ETA: Estimated Time of Arrival

NN: Nearest Neighbor

MSTs :minimum spanning trees

AOSP: Android Open Source Project

J2ME: Java Micro-Edition

APP: Application

INTRODUCTION

In computer science context awareness refers to the idea that computers can both sense, and react based on their environment.
Devices may have information about the circumstances under which they are able to operate based on rules and react accordingly. All modern Smartphone devices have GPS, Accelerometer components.

Many new possibilities arise:

- Location Aware Advertisements
- Location Aware Information
- Location Aware services (transportation, entertainment etc.)

Based on the above we decided to create an app whose goal would be to make user's life easier by helping them take care of everyday tasks in the minimum time. Hence, we came up with the idea of a location aware calendar, that the user would use as his daily agenda and would offer him the optimal route to visit all the day's destinations in the minimum time, thus saving time and money. Moreover we added extra functionality by changing the route according to our needs, view the address of every location on the map and implement messages on map's locations which will appear when the users arrive to them.

Chapter 1: ANDROID PLATFORM

1.1 General Description

Android is a Linux-based operating system for mobile devices such as smart phones and tablet computers, developed by Google in conjunction with the Open Handset Alliance. Android was initially developed by Android Inc, which Google financially backed and later purchased in 2005. The unveiling of the Android distribution in 2007 was announced with the founding of the Open Handset Alliance, a consortium of 86 hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices. Google releases the Android code as open-source, under the Apache License[28]. The Android Open Source Project (AOSP) is tasked with the maintenance and further development of Android. Additionally, Android has a large community of developers writing applications ("apps") that extend the functionality of devices. Developers write primarily in a customized version of Java, and apps can be downloaded from online stores such as Google Play (formerly Android Market), the app store run by Google, or third-party sites.[29]

1.2 Features

1.2.1 Handset layouts

The platform is adaptable to larger, VGA, 2D graphics library, 3D graphics library based on OpenGL ES 2.0 specifications, and traditional Smartphone layouts.

1.2.2 Storage

SQLite, a lightweight relational database, is used for data storage purposes.

1.2.3 Connectivity

Android supports connectivity technologies including GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC and WiMAX.

1.2.4 Messaging

SMS and MMS are available forms of messaging, including threaded text messaging and Android Cloud To Device Messaging (C2DM). The enhanced version of C2DM, Android Google Cloud Messaging (GCM), is also a part of Android Push Messaging service.

1.2.5 Multiple language support

Android supports multiple languages.

1.2.6 Web browser

The web browser available in Android is based on the open-source WebKit layout engine, coupled with Chrome's V8 JavaScript engine.

1.2.7 Java support

While most Android applications are written in Java, there is no Java Virtual Machine in the platform and Java byte code is not executed. Java classes are compiled into Dalvik executables and run on Dalvik, a specialized virtual machine designed for Android and optimized for battery-powered mobile devices with limited memory and CPU. J2ME (Java, Micro Edition) support is provided via third-party applications.

1.2.8 Media support

Android supports the following audio/video/still media formats: WebM, H.263, H.264 (in 3GP or MP4 container), MPEG-4 SP, AMR, AMR-WB (in 3GP container), AAC, HE-AAC (in MP4 or 3GP container), MP3, MIDI, Ogg Vorbis, FLAC, WAV, JPEG, PNG, GIF, BMP, WebP.

1.2.9 Streaming media support

RTP/RTSP streaming (3GPP PSS, ISMA), HTML progressive download (HTML5 <video> tag) are supported. Adobe Flash Streaming (RTMP) and HTTP Dynamic Streaming are supported by the Flash plugin. Apple HTTP Live Streaming is supported by RealPlayer for Android, and by the operating system in Android 3.0 (Honeycomb).

1.2.10 Additional hardware support

Android can use video/still cameras, touch screens, GPS, accelerometers, gyroscopes, barometers, magnetometers, dedicated gaming controls, proximity and pressure sensors, thermometers, accelerated 2D bit blits (with hardware orientation, scaling, pixel format conversion) and accelerated 3D graphics.

1.2.11 Multi-touch

Android has native support for multi-touch which was initially made available in handsets such as the HTC Hero. The feature was originally disabled at the kernel level (possibly to avoid infringing Apple's patents on touch-screen technology at the time). Google has since released an update for the Nexus One and the Motorola Droid which enables multi-touch natively.

1.2.12 Bluetooth

A2DP, AVRCP, sending files (OPP), accessing the phone book (PBAP), voice dialing and sending contacts between phones is supported. Keyboard, mouse and joystick (HID) support is available in Android 3.1+, and in earlier versions through manufacturer customizations and third-party applications.

1.2.13 Video calling

Android does not support native video calling, but some handsets have a customized version of the operating system that supports it, either via the UMTS network (like the Samsung Galaxy S) or over IP. Video calling through Google Talk is available in Android 2.3.4 and later versions. Gingerbread allows Nexus S to place Internet calls with a SIP account. This allows for enhanced VoIP dialing to other SIP accounts and even phone numbers. Skype 2.1 offers video calling in Android 2.3, including front camera support. Users with the Google+ android app can video chat with other Google+ users through hangouts.

1.2.14 Multitasking

Multitasking of applications, with unique handling of memory allocation, is available.

1.2.15 Voice based features

Google search through voice has been available since initial release. Voice actions for calling, texting, navigation, etc. are supported on Android 2.2 onwards.

1.2.16 Tethering

Android supports tethering, which allows a phone to be used as a wireless/wired Wi-Fi hotspot. Before Android 2.2 this was supported by third-party applications or manufacturer customizations.

1.2.17 Screen capture

Android supports capturing a screenshot by pressing the power and volume-down buttons at the same time. Prior to Android 4.0, the only methods of capturing a screenshot were through manufacturer and third-party customizations or otherwise by using a PC connection (DDMS developer's tool). These alternative methods are still available with the latest Android.

1.2.18 External storage

Most Android devices include microSD slot and can read microSD cards formatted with FAT32, Ext3 or Ext4 file system. To allow use of high-capacity storage media such as USB flash drives and USB HDDs, many Android tablets also include USB 'A' receptacle. Storage formatted with FAT32 is handled by Linux Kernel VFAT driver, while third party solutions are required to handle other popular file systems such as NTFS, HFS Plus and exFAT.

1.3 Android version history

The version history of the Android operating system began with the release of the Android beta in November 2007. The first commercial version, Android 1.0, was released in September 2008. Since its original release it has seen a number of updates to its base operating system. These updates typically fix bugs and add new features. Since April 2009, each Android version has been developed under a codename based on a dessert or sweet treat. These versions have been released in alphabetical order:[30]

- 1 Android beta
- 2 Android 1.0 Astro
- 3 Android 1.1 Bender
- 4 Android 1.5 Cupcake
- 5 Android 1.6 Donut
- 6 Android 2.0/2.1 Eclair
- 7 Android 2.2.x Froyo
- 8 Android 2.3.x Gingerbread
- 9 Android 3.x Honeycomb
- 10 Android 4.0.x Ice Cream Sandwich
- 11 Android 4.1.x Jelly Bean

1.3.1 Gingerbread[16]

Our application was developed for the Gingerbread version therefore we focus on this version from now on. This version brings the following improvements from the Froyo version for both consumers and developers.

1.3.1.1 UI refinements for simplicity and speed

The user interface is refined in many ways across the system, making it easier to learn, faster to use, and more power-efficient. A simplified visual theme of colors against black brings vividness and contrast to the notification bar, menus, and other parts of the UI. Changes in menus and settings make it easier for the user to navigate and control the features of the system and device.

1.3.1.2 Faster, more intuitive text input

The Android soft keyboard is redesigned and optimized for faster text input and editing. The keys themselves are reshaped and repositioned for improved targeting, making them easier to see and press accurately, even at high speeds. The keyboard also displays the current character and dictionary suggestions in a larger, more vivid style that is easier to read.

The keyboard adds the capability to correct entered words from suggestions in the dictionary. As the user selects a word already entered, the keyboard displays suggestions that the user can choose from, to replace the selection. The user can also switch to voice input mode to replace the selection. Smart suggestions let the user accept a suggestion and then return to correct it later, if needed, from the original set of suggestions.

New multitouch key-chording lets the user quickly enter numbers and symbols by pressing Shift+<letter> and ?123+<symbol>, without needing to manually switch input modes. From certain keys, users can also access a popup menu of accented characters, numbers, and symbols by holding the key and sliding to select a character.

1.3.1.3 One-touch word selection and copy/paste

When entering text or viewing a web page, the user can quickly select a word by press-hold, then copy to the clipboard and paste. Pressing on a word enters a free-selection mode — the user can adjust the selection area as needed by dragging a set of bounding arrows to new positions, then copy the bounded area by pressing anywhere in the selection area. For text entry, the user can slide-press to enter a cursor mode, then reposition the cursor easily and accurately by dragging the cursor arrow. With both the selection and cursor modes, no use of a trackball is needed.

1.3.1.4 Improved power management

The Android system takes a more active role in managing apps that are keeping the device awake for too long or that are consuming CPU while running in the background. By managing such apps — closing them if appropriate — the system helps ensure best possible performance and maximum battery life.

The system also gives the user more visibility over the power being consumed by system components and running apps. The Application settings provide an accurate overview of how the battery is being used, with details of the usage and relative power consumed by each component or application.

1.3.1.5 Control over applications

A shortcut to the Manage Applications control now appears in the Options Menu in the Home screen and Launcher, making it much easier to check and manage application activity. Once the user enters Manage Applications, a new Running tab displays a list of active applications and the storage and memory being used by each. The user can read further details about each application and if necessary stop an application or report feedback to its developer.

1.3.1.6 New ways of communicating, organizing

An updated set of standard applications lets the user take new approaches to managing information and relationships.

1.3.1.7 Internet calling

The user can make voice calls over the internet to other users who have SIP accounts. The user can add an internet calling number (a SIP address) to any Contact and can initiate a call from Quick Contact or Dialer. To use internet calling, the user must create an account at the SIP provider of their choice — SIP accounts are not provided as part of the internet calling feature. Additionally, support for the platform's SIP and internet calling features on specific devices is determined by their manufacturers and associated carriers.

1.3.1.8 Near-field communications

An NFC Reader application lets the user read and interact with near-field communication (NFC) tags. For example, the user can "touch" or "swipe" an NFC tag that might be embedded in a poster, sticker, or advertisement, then act on the data read from the tag. A typical use would be to read a tag at a restaurant, store, or event and then rate or register by jumping to a web site whose URL is included in the tag data. NFC communication relies on wireless technology in the device hardware, so support for the platform's NFC features on specific devices is determined by their manufacturers.

1.3.1.9 Downloads management

The Downloads application gives the user easy access to any file downloaded from the browser, email, or another application. Downloads is built on a completely new download manager facility in the system that any other applications can use, to more easily manage and store their downloads.

1.3.1.10 Camera

The application now lets the user access multiple cameras on the device, including a front-facing camera, if available.

New Developer Features

Android 2.3 delivers a variety of features and APIs that let developers bring new types of applications to the Android platform.

- Enhancements for gaming
- New forms of communication
- Rich multimedia

Enhancements for gaming

1.3.1.11 Performance

Android 2.3 includes a variety of improvements across the system that makes common operations faster and more efficient for all applications. Of particular interest to game developers are:

Concurrent garbage collector — The Dalivik VM introduces a new, concurrent garbage collector that minimizes application pauses, helping to ensure smoother animation and increased responsiveness in games and similar applications.

Faster event distribution — the platform now handles touch and keyboard events faster and more efficiently, minimizing CPU utilization during event distribution. The changes improve responsiveness for all applications, but especially benefit games that use touch events in combination with 3D graphics or other CPU-intensive operations.

Updated video drivers — the platform uses updated third-party video drivers that improve the efficiency of OpenGL ES operations, for faster overall 3D graphics performance.

1.3.1.12 Native input and sensor events

Applications that use native code can now receive and process input and sensor events directly in their native code, which dramatically improves efficiency and responsiveness.

Native libraries exposed by the platform let applications handle the same types of input events as those available through the framework. Applications can receive events from all supported sensor types and can enable/disable specific sensors and manage event delivery rate and queuing.

1.3.1.13 Gyroscope and other new sensors, for improved 3D motion processing

Android 2.3 adds API support for several new sensor types, including gyroscope, rotation vector, linear acceleration, gravity, and barometer sensors. Applications can use the new sensors in combination with any other sensors available on the device, to track three-dimensional device motion and orientation change with high precision and accuracy. For example, a game application could use readings from a gyroscope and accelerometer on the device to recognize complex user gestures and motions, such as tilt, spin, thrust, and slice.

1.3.1.14 Open API for native audio

The platform provides a software implementation of Khronos OpenSL ES, a standard API that gives applications access to powerful audio controls and effects from native code. Applications can use the API to manage audio devices and control audio input, output, and processing directly from native code.

1.3.1.15 Native graphics management

The platform provides an interface to its Khronos EGL library, which lets applications manage graphics contexts and create and manage OpenGL ES textures and surfaces from native code.

1.3.1.16 Native access to Activity lifecycle, window management

Native applications can declare a new type of Activity class, NativeActivity, whose lifecycle callbacks are implemented directly in native code. The NativeActivity and its underlying native code run in the system just as do other Activities — they run in the application's system process and execute on the application's main UI thread, and they receive the same lifecycle callbacks as do other Activities.

1.3.1.17 Native access to assets, storage

Applications can now access a native Asset Manager API to retrieve application assets directly from native code without needing to go through JNI. If the assets are compressed, the platform does streaming decompression as the application reads the asset data. There is no longer a limit on the size of compressed .apk assets that can be read.

Additionally, applications can access a native Storage Manager API to work directly with OBB files downloaded and managed by the system.

1.3.1.18 Robust native development environment

The Android NDK (Native Development Kit) provides a complete set of tools, toolchains, and libraries for developing applications that use the rich native environment offered by the Android 2.3 platform. For more information or to download the NDK, please see the Android NDK page.

New forms of communication

1.3.1.19 Internet telephony

Developers can now add SIP-based internet telephony features to their applications. Android 2.3 includes a full SIP protocol stack and integrated call management services that let applications easily set up outgoing and incoming voice calls, without having to manage sessions, transport-level communication, or audio record or playback directly.

Support for the platform's SIP and internet calling features on specific devices is determined by their manufacturers and associated carriers.

1.3.1.20 Near Field Communications (NFC)

The platform's support for Near Field Communications (NFC) lets developers get started creating a whole new class of applications for Android. Developers can create new applications that offer proximity-based information and services to users, organizations, merchants, and advertisers.

Using the NFC API, applications can read and respond to NFC tags "discovered" as the user "touches" an NFC-enabled device to elements embedded in stickers, smart posters, and even other devices. When a tag of interest is collected, applications can respond to the tag, read messages from it, and then store the messages, prompting the user as needed.

Starting from Android 2.3.3, applications can also write to tags and set up peer-to-peer connections with other NFC devices.

NFC communication relies on wireless technology in the device hardware, so support for the platform's NFC features on specific devices is determined by their manufacturers.

Rich multimedia

1.3.1.21 Mixable audio effects

A new audio effects API lets developers easily create rich audio environments by adding equalization, bass boost, headphone virtualization (widened soundstage), and reverb to audio tracks and sounds. Developers can mix multiple audio effects in a local track or apply effects globally, across multiple tracks.

1.3.1.22 Support for new media formats

The platform now offers built-in support for the VP8 open video compression format and the WebM open container format. The platform also adds support for AAC encoding and AMR wideband encoding (in software), so that applications can capture higher quality audio than narrowband.

1.3.1.23 Access to multiple cameras

The Camera API now lets developers access any cameras that are available on a device, including a front-facing camera. Applications can query the platform for the number of cameras on the device and their types and characteristics, then open the camera needed. For example, a video chat application might want to access a front-facing camera that offers lower-resolution, while a photo application might prefer a back-facing camera that offers higher-resolution.

Chapter 2: CREATING AN ANDROID PROJECT

An Android project contains all the files that comprise the source code for your Android app. The Android SDK tools make it easy to start a new Android project with a set of default project directories and files.

Step 1: Installing the Android SDK

2.1 What is the android SDK

The android SDK is a software development kit that enables developers to create applications for the Android platform. The Android SDK includes sample projects with source code, development tools, an emulator, and required libraries to build Android applications. Applications are written using the Java programming language and run on Dalvik, a custom virtual machine designed for embedded use which runs on top of a Linux kernel.

2.2 Exploring the SDK[14]

The Android SDK is composed of modular packages that you can download separately using the Android SDK Manager. There are several different packages available for the Android SDK. The table below describes most of the available packages and where they're located once you download them.

Available Packages

SDK Tools: Contains tools for debugging and testing, plus other utilities that are required to develop an app.

SDK Platform-tools: Contains platform-dependent tools for developing and debugging your application. These tools support the latest features of the Android platform and are typically updated only when a new platform becomes available. These tools are always backward compatible with older platforms.

Documentation: An offline copy of the latest documentation for the Android platform APIs.

SDK Platform: There's one SDK Platform available for each version of Android. It includes an android.jar file with a fully compliant Android library. In order to build an Android app, you must specify an SDK platform as your build target.

System Images: Each platform version offers one or more different system images (such as for ARM and x86). The Android emulator requires a system image to operate.

Sources for Android SDK: A copy of the Android platform source code that's useful for stepping through the code while debugging the app.

Samples for SDK: A collection of sample apps that demonstrate a variety of the platform APIs. These are a great resource to browse Android app code.

Google APIs: An SDK add-on that provides both a platform you can use to develop an app using special Google APIs and a system image for the emulator so you can test the app using the Google APIs.

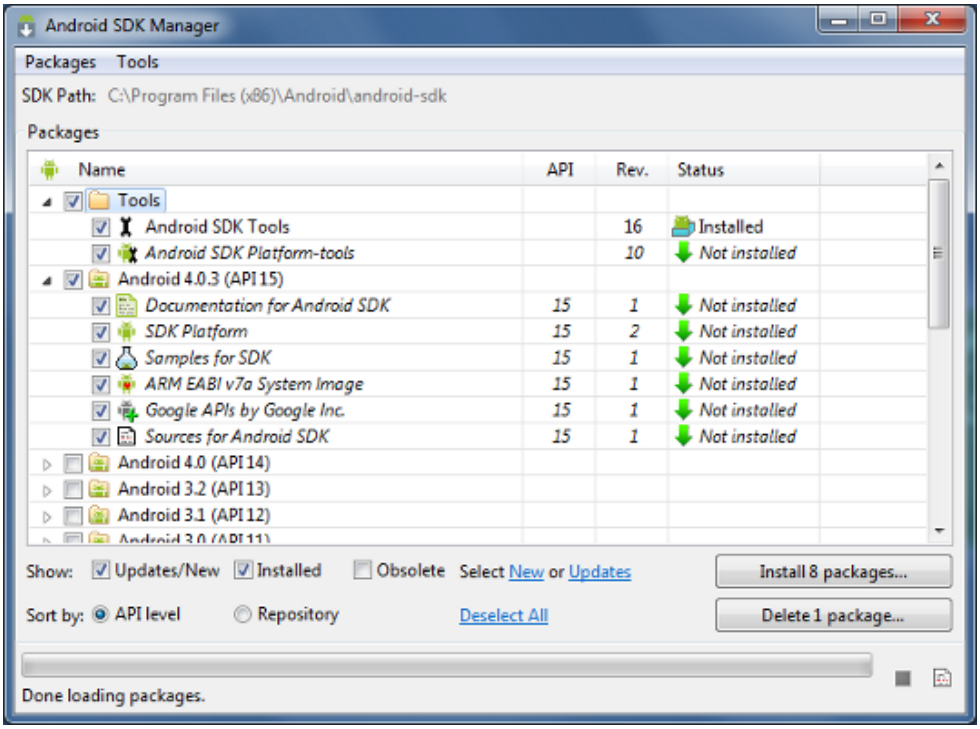
Android Support: A static library you can include in your app sources in order to use powerful APIs that aren't available in the standard platform.

Google Play Billing: Provides the static libraries and samples that allow you to integrate billing services in your app with Google Play.

Google Play Licensing: Provides the static libraries and samples that allow you to perform license verification for your app when distributing with Google Play.

2.3 Adding Platforms and Packages[32]

The Android SDK separates tools, platforms, and other components into packages we can download using the Android SDK Manager. When you open the Android SDK Manager, it automatically selects a set of recommended packages. Simply click Install to install the recommended packages. The Android SDK Manager installs the selected packages into your Android SDK environment.



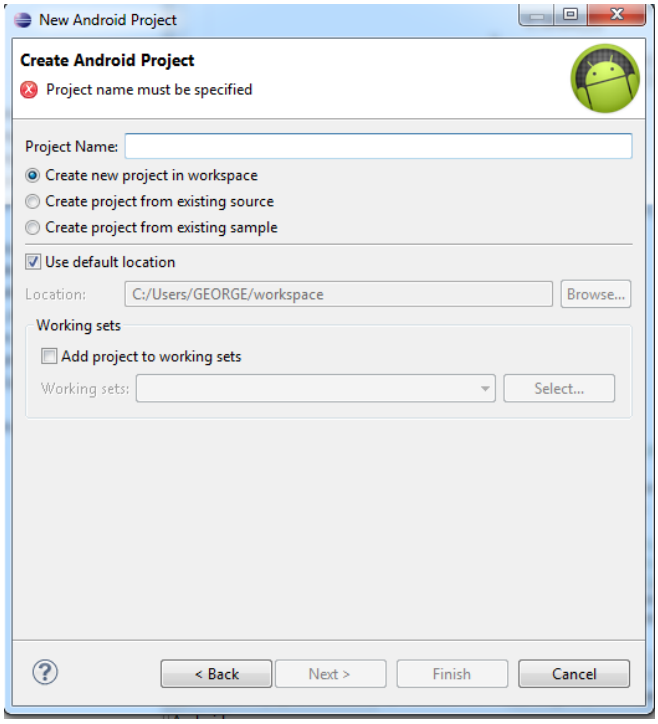
2.4 Installing the Eclipse Plugin[33]

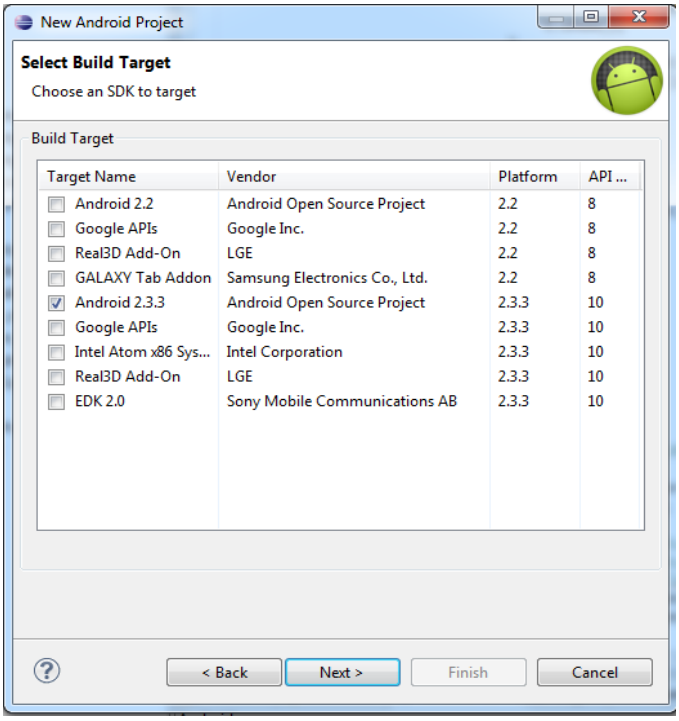
Our app was developed using the eclipse Indigo. Because of that we had to install a custom plug-in that android offers for Eclipse IDE, called Android Development Tools(ADT). This plug-in is designed to give a powerful, integrated environment in which to develop Android apps. It extends the capabilities of Eclipse to quickly set up new Android projects, build an app UI, debug the app, and export signed (or unsigned) app packages (APKs) for distribution.

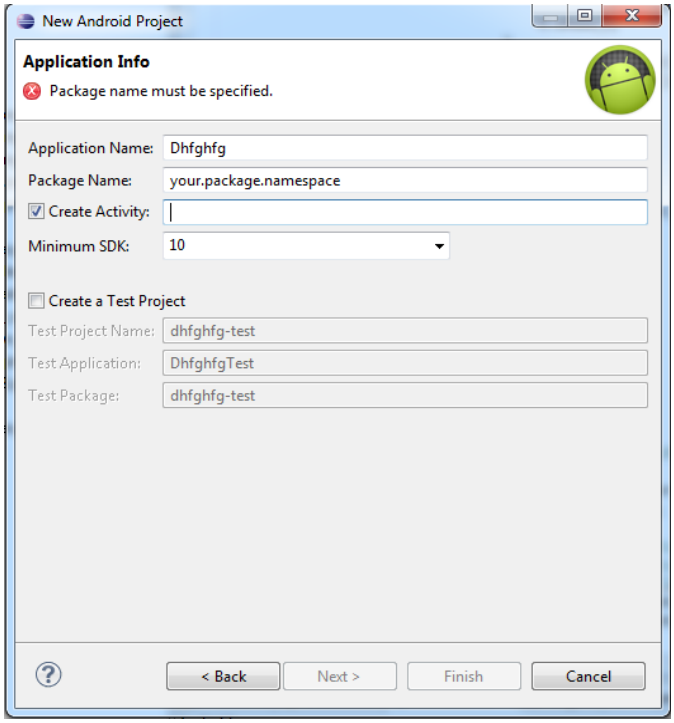
STEP 2: Create a Project with Eclipse[34]

2.5 Using the Eclipse Indigo

- In Eclipse, click New Android App Project in the toolbar.
- Application Name is the app name that appears to users.
- Project Name is the name of project directory and the name visible in Eclipse.
- Package Name is the package namespace for the app. The package name must be unique across all packages installed on the Android system.
- Build SDK is the platform version against which you will compile your app. By default, this is set to the latest version of Android available in your SDK.
- Minimum Required SDK is the lowest version of Android that your app supports.
- The next screen provides tools to create a launcher icon for the app.
- Next you select an activity template from which to begin building your app.
- Finally, leave all the details for the activity in their default state and click Finish.


New Android Project
Create Android Project
Project name must be specified
Project Name:
Create new project in workspace
Create project from existing source
Create project from existing sample
Use default location
Location:
C:/Users/GEORGE/workspace
Browse...
Working sets
Add project to working sets
Working sets:
Select...
< Back
Next >
Finish
Cancel


New Android Project
Select Build Target
Choose an SDK to target
Build Target
Target Name
Vendor
Platform
API ...
Android 2.2
Android Open Source Project
2.2
8
Google APIs
Google Inc.
2.2
8
Real3D Add-On
LGE
2.2
8
GALAXY Tab Addon
Samsung Electronics Co., Ltd.
2.2
8
Android 2.3.3
Android Open Source Project
2.3.3
10
Google APIs
Google Inc.
2.3.3
10
Intel Atom x86 Sys...
Intel Corporation
2.3.3
10
Real3D Add-On
LGE
2.3.3
10
EDK 2.0
Sony Mobile Communications AB
2.3.3
10
< Back
Next >
Finish
Cancel


New Android Project
Application Info
Package name must be specified.
Application Name:
Dhfghfg
Package Name:
your.package.namespace
Create Activity:
Minimum SDK:
10
Create a Test Project
Test Project Name:
dhfghfg-test
Test Application:
DhfghfgTest
Test Package:
dhfghfg-test
< Back
Next >
Finish
Cancel

STEP 3: Running the Application [35]

The android project that I have created contains the following directories and files whose knowledge is essential for the development:

2.6 Directories and files

- **AndroidManifest.xml**

The manifest file describes the fundamental characteristics of the app and defines each of its components.

- **src/**

Directory for the app's main source files. By default, it includes an Activity class that runs when the app is launched using the app icon.

- **res/**

Contains several sub-directories for app resources.

❖ **drawable-hdpi/**

Directory for drawable objects (such as bitmaps) that are designed for high-density (hdpi) screens. Other drawable directories contain assets designed for other screen densities.

❖ **layout/**

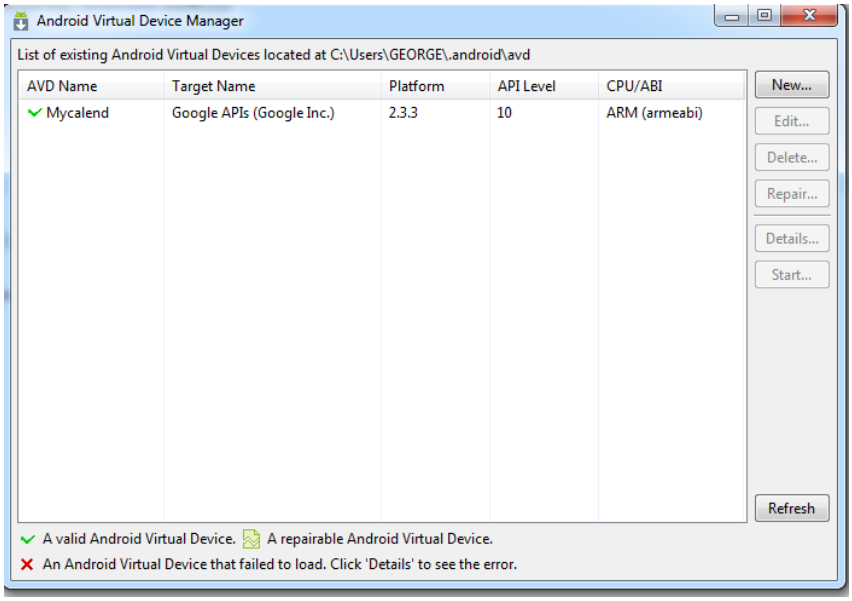
Directory for files that define the app's user interface.

❖ **values/**

Directory for other various XML files that contain a collection of resources, such as string and color definitions.

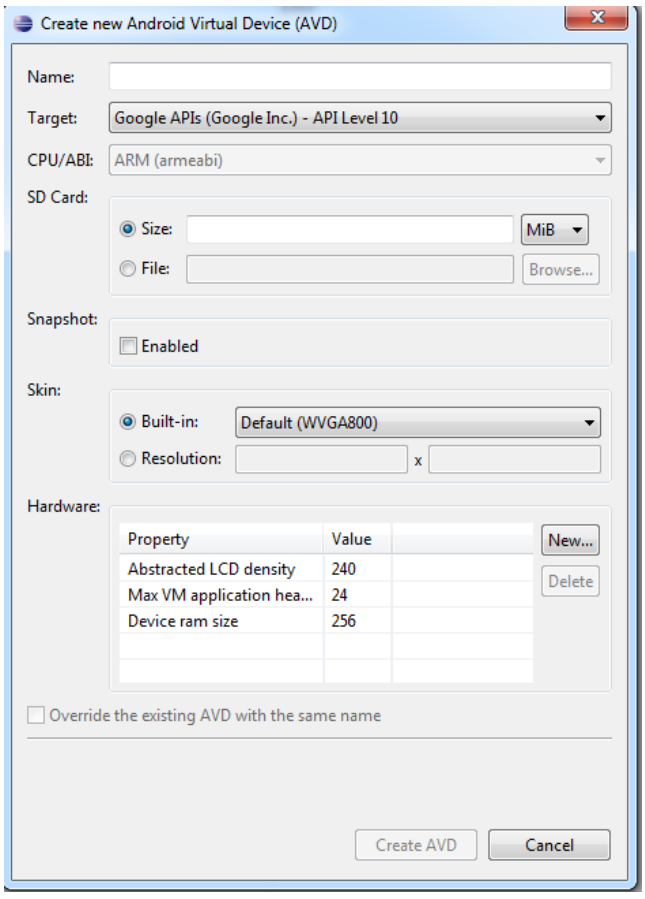
2.7 Available options for running the application

- Run on a Real Device
- Run on the Emulator



2.7.1 Creating an android Virtual Device (AVD)

To create an Android Virtual Device (AVD) we click "new" and at the next window you select as target the primary android version that our app is going to support. In our case that is Google APIs –API Level 10 that includes support for **Google maps**. The next option that we should select is *the resolution support* which in our case is the default WVGA800 resolution or 800*480 pixels. The other options are left at the default selection. Then, we click *Create AVD.*



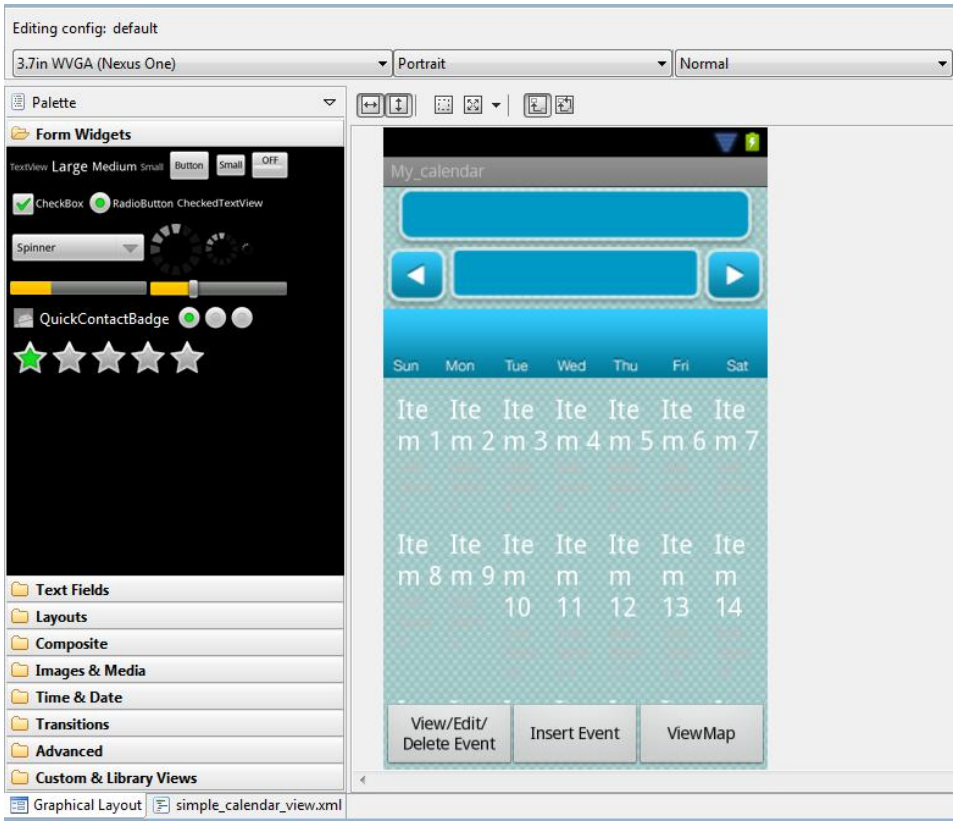
Chapter 3: DESIGNING AGENDA-ROUTE PLANNER

3.1 User Interface [36]

The graphical user interface for an Android app is built using a hierarchy of View and ViewGroup objects. View objects are usually UI widgets such as buttons or text fields and ViewGroup objects are invisible view containers that define how the child views are laid out, such as in a grid or a vertical list. Android provides an XML vocabulary that corresponds to the subclasses of View and ViewGroup so you can define UI in XML using a hierarchy of UI elements.

3.1.1 Supporting Different Screens [37]

Android categorizes device screens using two general properties: size and density. There are four generalized sizes: small, normal, large, xlarge, and four generalized densities: low (ldpi), medium (mdpi), high (hdpi), extra high (xhdpi). Our application was designed for 3.7 inches screens with resolution of 800*480, although it can work in other screen sizes because the main factor here is the resolution. For example the phone we used to test the app has a 3.8 inch screen.



3.1.2 Layouts [20]

A layout defines the visual structure for a user interface, such as the UI for an activity or app widget. We can declare a layout in two ways:

1. Declare UI elements in XML. Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
2. Instantiate layout elements at runtime. The application can create View and ViewGroup objects (and manipulate their properties).

The Android framework gives you the flexibility to use either or both of these methods for declaring and managing your application. In our application both methods were used, the first to create the MapView and add buttons to the main screen and the second to populate the date boxes.

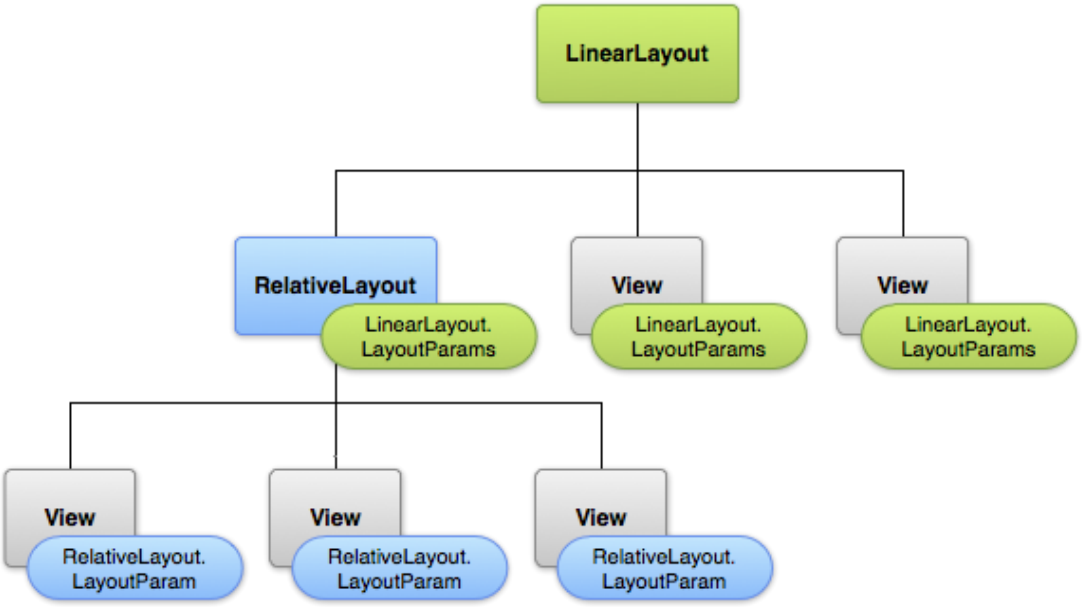
The advantage to declaring our UI in XML is that it enables us to better separate the presentation of our application from the code that controls its behavior. Our UI descriptions are external to the application code, which means that we can modify or adapt it without having to modify our source code and recompile. Additionally, declaring the layout in XML makes it easier to visualize the structure of our UI, so it's easier to debug problems.

In general, the XML vocabulary for declaring UI elements closely follows the structure and naming of the classes and methods, where element names correspond to class names and attribute names correspond to methods.

3.1.2.1 Layout Parameters

XML layout attributes named layout_something define layout parameters for the View that is appropriate for the ViewGroup in which it resides. Every ViewGroup class implements a nested class that extends ViewGroup.LayoutParams. This subclass contains property types that define the size and position for each child view, as appropriate for the

view group. As shown below, the parent view group defines layout parameters for each child view (including the child view group).



Every LayoutParams subclass has its own syntax for setting values. Each child element must define LayoutParams that are appropriate for its parent, though it may also define different LayoutParams for its own children. All view groups include a width and height (layout_width and layout_height), and each view is required to define them. Many LayoutParams also include optional margins and borders.

These constants are used to set the width or height:

wrap_content: tells your view to size itself to the dimensions required by its content

fill_parent :tells your view to become as big as its parent view group will allow.

3.1.2.2 Create a Relative Layout

In our application we used a relative layout as the default layout of the app. RelativeLayout is a view group that displays child views in relative positions. The position of each view can be specified as relative to sibling elements (such as to the left-of or below another view) or in

positions relative to the parent RelativeLayout area (such as aligned to the bottom, left of center).

A RelativeLayout is a very powerful utility for designing a user interface because it can eliminate nested view groups and keep layout hierarchy flat, which improves performance.

3.1.2.3 Positioning Views

RelativeLayout lets child views specify their position relative to the parent view or to each other (specified by ID). So you can align two elements by right border, or make one below another, centered in the screen, centered left, and so on. By default, all child views are drawn at the top-left of the layout, so you must define the position of each view using the various layout properties. Some of the many layout properties available to views in a RelativeLayout include:

- android:layout_alignParentTop

  If "true", makes the top edge of this view match the top edge of the parent.

- android:layout_centerVertical

  If "true", centers this child vertically within its parent.

- android:layout_below

  Positions the top edge of this view below the view specified with a resource ID.

- android:layout_toRightOf

  Positions the left edge of this view to the right of the view specified with a resource ID.

The value for each layout property is either a Boolean to enable a layout position relative to the parent RelativeLayout or an ID that references another view in the layout against which the view should be positioned.

3.1.2.4 Attributes

Every View and ViewGroup object supports their own variety of XML attributes. Some attributes are specific to a View object, but these attributes are also inherited by any View objects that may extend this class. Some are common to all View objects, because they are inherited from the root View class. Other attributes are considered "layout parameters," i.e., attributes that describe certain layout orientations of the View object, as defined by that object's parent ViewGroup object.

3.1.2.4.1 Add a Text Field

To create a user-editable text field, you add an <EditText> element inside the <LinearLayout>.Like every View object, I must define certain XML attributes to specify the EditText object's properties.

```
<EditText android:id="@+id/edit_message"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="@string/edit_message" />
```

Regarding attributes:

android:id="@+id/edit_message"

This provides a unique identifier for the view, which we can use to reference the object from our app code, such as to read and manipulate the object. The at sign (@) is required when we are referring to any resource object from XML. It is followed by the resource type (id in this case), a slash, then the resource name (edit_message). The plus sign (+) before the resource type is needed only when you're defining a resource ID for the first time. When you compile the app, the SDK tools use the ID name to create a new resource ID in your project's gen/R.java file that refers to the EditText element. Once the resource ID is declared once this way, other references to the ID do not need the plus sign. Using the plus sign is necessary only when specifying a new resource ID and not needed for concrete resources such as strings or layouts.

- **android:layout_width="wrap_content"**
- **android:layout_height="wrap_content"**

Instead of using specific sizes for the width and height, the "wrap_content" value specifies that the view should be only as big as needed to fit the contents of the view. If you were to instead use "match_parent", then the EditText element would fill the screen, because it would match the size of the parent LinearLayout.

android:hint="@string/edit_message"

This is a default string to display when the text field is empty. Instead of using a hard-coded string as the value, the "@string/edit_message" value refers to a string resource defined in a separate file. Because this refers to a concrete resource (not just an identifier), it does not need the plus sign.

3.1.2.4.2 Add String Resources

When you need to add text in the user interface, you should always specify each string as a resource. String resources allow you to manage all UI text in a single location, which makes it easier to find and update text. Externalizing the strings also allows you to localize your app to different languages by providing alternative definitions for each string resource. By default, Android project includes a string resource file at res/values/strings.xml. A strings.xml looks like this:

```
<?xml version="1.0" encoding="utf-8"?>

<resources>

   <string name="app_name">My First App</string>

   <string name="edit_message">Enter a message</string>

   <string name="button_send">Send</string>

   <string name="menu_settings">Settings</string>

   <string name="title_activity_main">MainActivity</string>

</resources>
```

3.1.2.4.3 Add a Button

In order to declare a button to the xml file the code should look like this:

```
<Button
      android:id="@+id/selectedDayMonthYear"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"
      android:layout_gravity="center"
      android:textColor="#FFFFFF">
   </Button>
```

All of the above can be combined to create an xml file that designs applications layout.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/RelativeLayout1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@drawable/bg"
    android:gravity="center|left|right"
    android:orientation="vertical" >

    <Button
        android:id="@+id/selectedDayMonthYear"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_gravity="center"
        android:background="@drawable/calendar_top_header"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:textColor="#FFFFFF" >
    </Button>

    <ImageView
        android:id="@+id/prevMonth"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/selectedDayMonthYear"
        android:src="@drawable/calendar_left_arrow_selector" >
    </ImageView>

    <Button
        android:id="@+id/currentMonth"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/prevMonth"
        android:layout_alignTop="@+id/prevMonth"
        android:layout_toLeftOf="@+id/nextMonth"
        android:layout_toRightOf="@+id/prevMonth"
        android:layout_weight="0.6"
        android:background="@drawable/calendar_bar"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:textColor="#FFFFFF" >
```

Graphical Layout | simple_calendar_view.xml

Chapter 4: PROGRAMMING AGENDA-ROUTE PLANNER

4.1 Supporting Different Platform Versions [26]

4.1.1 Specify Minimum and Target API Levels

The **AndroidManifest.xml** file describes details about the app and identifies which versions of Android it supports. Specifically, the minSdkVersion and targetSdkVersion attributes for the <uses-sdk element identify the lowest API level with which the app is compatible and the highest API level against which we have designed and tested the app. In our case the minimum API Level is API 8 (Android 2.2 Froyo) and the target API Level is API 10 (Android 2.3 Gingerbread).

4.1.2 Security Architecture (Permissions)

A central design point of the Android security architecture is that no application, by default, has permission to perform any operations that would adversely impact other applications, the operating system, or the user. Because Android sandboxes applications from each other, applications must explicitly share resources and data. They do this by declaring the permissions they need for additional capabilities not provided by the basic sandbox. Applications statically declare the permissions they require, and the Android system prompts the user for consent at the time the application is installed.

The structure of a permission declared in the manifest is the following:

- **syntax:**

  <uses-permission android:name="string" />

- **description:**

Requests a permission that the application must be granted in order for it to operate correctly.

- **attributes:**

android:name

The name of the permission. It can be a permission defined by the application with the <permission> element, a permission defined by another application, or one of the standard system permissions, such as "android.permission.CAMERA" or "android.permission.READ_CONTACTS". As these examples show, a permission name typically includes the package name as a prefix.

4.1.2.1 Permissions Used in the application

- ***android:name="android.permission.INTERNET"/>***

  Allows applications to open network sockets.

- ***android:name="android.permission.ACCESS_FINE_LOCATION"/>***

  Allows an application to access fine (e.g., GPS) location

- ***android:name="android.permission.ACCESS_COARSE_LOCATION"/>***

  Allows an application to access coarse (e.g., Cell-ID, WiFi) location

- ***android:name="android.permission.READ_CALENDAR"/>***

  Allows an application to read the user's calendar data.

- ***android:name="android.permission.ACCESS_NETWORK_STATE"/>***

  Allows applications to access information about networks

- ***android:name="android.permission.ACCESS_WIFI_STATE"/>***

  Allows applications to access information about Wi-Fi networks

4.2 Activities [2]

An Activity is an application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map. Each activity is given a window in which to draw its user interface. The window typically fills the screen, but may be smaller than the screen and float on top of other windows.

An application usually consists of multiple activities that are loosely bound to each other. Typically, one activity in an application is specified as the "main" activity, which is presented to the user when launching the application for the first time. Each activity can then start another activity in order to perform different actions. Each time a new activity starts, the previous activity is stopped, but the system preserves the activity in a stack (the "back stack"). When a new activity starts, it is pushed onto the back stack and takes user focus. The back stack abides to the basic "last in, first out" stack mechanism, so, when the user is done with the current activity and presses the Back button, it is popped from the stack (and destroyed) and the previous activity resumes.

When an activity is stopped because a new activity starts, it is notified of this change in state through the activity's lifecycle callback methods. There are several callback methods that an activity might receive, due to a change in its state—whether the system is creating it, stopping it, resuming it, or destroying it—and each callback provides the opportunity to perform specific work that's appropriate to that state change. For instance, when stopped, our activity should release any large objects, such as network or database connections. When the activity resumes, we can reacquire the necessary resources and resume actions that were interrupted. These state transitions are all part of the activity lifecycle.

4.2.1 Creating an Activity

To create an activity, we must create a subclass of Activity. In our subclass, you need to implement callback methods that the system calls when the activity transitions between various states of its lifecycle, such as when the activity is being created, stopped, resumed, or destroyed. The two most important callback methods are:

- **onCreate()**

The system calls this when creating our activity. Within our implementation, we should initialize the essential components of our activity. Most importantly, this is where we need to call setContentView() to define the layout for the activity's user interface.

- **onPause()**

The system calls this method as the first indication that the user is leaving our activity (though it does not always mean the activity is being destroyed). This is usually where we should commit any changes that should persist beyond the current user session (because the user might not come back). Moreover, by pausing the activity we conserve data and battery.

4.2.2 Declaring the activity in the manifest

We must declare our activity in the manifest file in order for it to be accessible to the system. To declare the activity, we add in the manifest file an <activity> element as a child of the <application> element. The structure of this declaration is the following

```
<manifest ... >
  <application ... >
    <activity android:name=".ExampleActivity" />
    ...
  </application ... >
  ...</manifest >
```

4.2.3 Using intent filters

An <activity> element can also specify various intent filters—using the <intent-filter> element—in order to declare how other application components may activate it. When we create a new application using the Android SDK tools, the stub activity that's created automatically includes an intent filter that declares the activity responses to the "main" action and should be placed in the "launcher" category. The intent filter looks like this:

```
<activity android:name=".ExampleActivity"
android:icon="@drawable/app_icon">
   <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
</activity>
```

The <action> element specifies that this is the "main" entry point to the application. The <category> element specifies that this activity should be listed in the system's application launcher (to allow users to launch this activity).

If we want our application to be self-contained and not allow other applications to activate its activities, then we don't need any other intent filters. Only one activity should have the "main" action and "launcher" category, as in the previous example. Activities that we don't want to make available to other applications should have no intent filters.

However, if you want your activity to respond to implicit intents that are delivered from other applications, then you must define additional intent filters for your activity. For each type of intent to which you want to respond, you must include an <intent-filter> that includes an <action> element and, optionally, a <category> element and/or a

‹data› element. These elements specify the type of intent to which our activity can respond.

In the following image we can see the application's manifest file that implements everything that is mentioned so far in this section.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.examples"
      android:versionCode="1"
      android:versionName="1.0">
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-sdk android:minSdkVersion="8" android:targetSdkVersion="10"/>

<uses-permission android:name="android.permission.READ_CALENDAR"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
    <application android:icon="@drawable/icon"

        android:label="@string/app_name">
              <uses-library android:name="com.google.android.maps"/>

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

    </application>


</manifest>
```

Manifest | Application | Permissions | Instrumentation | AndroidManifest.xml

4.2.4 Managing the Activity Lifecycle

The lifecycle of an activity is directly affected by its association with other activities, its task and back stack. An activity can exist in essentially three states:

1. **Resumed**

The activity is in the foreground of the screen and has the user's focus.

2. **Paused**

Another activity is in the foreground and has focus, but this one is still visible. That is, another activity is visible on top of this one and that activity is partially transparent or doesn't cover the entire screen. A paused activity is completely alive (the Activity object is retained in memory, it maintains all state and member information, and remains attached to the window manager), but can be killed by the system in extremely low memory situations.

3. **Stopped**

The activity is completely obscured by another activity (the activity is now in the "background"). A stopped activity is also still alive (the Activity object is retained in memory, it maintains all state and member information, but is not attached to the window manager). However, it is no longer visible to the user and it can be killed by the system when memory is needed elsewhere.

If an activity is paused or stopped, the system can drop it from memory either by asking it to finish (calling its finish() method), or simply killing its process. When the activity is opened again (after being finished or killed), it must be created all over.

4.2.5 Implementing the lifecycle callbacks

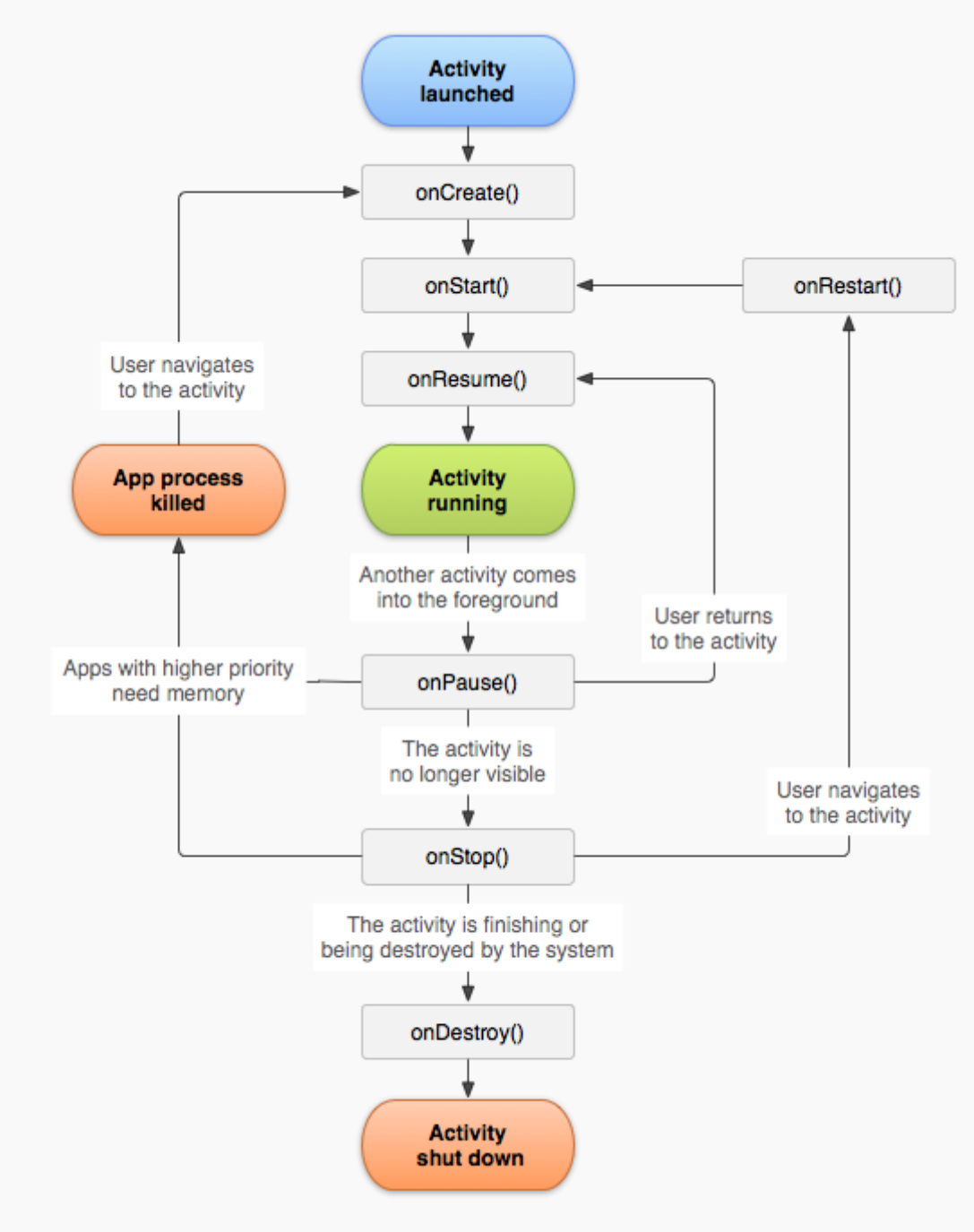
When an activity transitions into and out of the different states described above, it is notified through various callback methods. All of the callback methods are hooks that we can override to do appropriate work when the state of our activity changes. The callbacks methods available are the following:

- *onCreate()*
- *onStart()*
- *onResume()*
- *onPause()*
- *onStop()*
- *onDestroy()*

Taken together, these methods define the entire lifecycle of an activity. By implementing these methods, we can monitor three nested loops in the activity lifecycle:

- *The entire lifetime of an activity happens between the call to onCreate() and the call to onDestroy(). Our activity should perform setup of "global" state (such as defining layout) in onCreate(), and release all remaining resources in onDestroy().*
- *The visible lifetime of an activity happens between the call to onStart() and the call to onStop(). During this time, the user can see the activity on-screen and interact with it.*
- *The foreground lifetime of an activity happens between the call to onResume() and the call to onPause(). During this time, the activity is in front of all other activities on screen and has user input focus. An activity can frequently transition in and out of the foreground.*

The following diagram illustrates these loops and the paths an activity might take between states. The rectangles represent the callback methods that can be implemented to perform operations when the activity transitions between states.



The lifecycle callback methods are listed in the following table, which describes each of the callback methods in more detail and locates each one within the activity's overall lifecycle, including whether the system can kill the activity after the callback method completes.

Method	Description	Killable after?	Next
`onCreate()`	Called when the activity is first created. This is where you should do all of your normal static set up — create views, bind data to lists, and so on. This method is passed a Bundle object containing the activity's previous state, if that state was captured (see Saving Activity State, later). Always followed by `onStart()`.	No	`onStart()`
`onRestart()`	Called after the activity has been stopped, just prior to it being started again. Always followed by `onStart()`	No	`onStart()`

`onStart()`	Called just before the activity becomes visible to the user.
	Followed by `onResume()` if the activity comes to the foreground, or `onStop()` if it becomes hidden.
		No	`onResume()`
			or
			`onStop()`

`onResume()`	Called just before the activity starts interacting with the user. At this point the activity is at the top of the activity stack, with user input going to it. Always followed by `onPause()`.	No	`onPause()`

`onPause()`	Called when the system is about to start resuming another activity. This method is typically used to commit unsaved changes to persistent data, stop animations and other things that may be consuming CPU, and so on. It should do whatever it does very quickly, because the next activity will not be resumed until it returns.
	Followed either by `onResume()` if the activity returns back to the front, or by `onStop()` if it becomes invisible to the user.
		Yes	`onResume()`
			or
			`onStop()`

`onStop()`	Called when the activity is no longer visible to the user. This may happen because it is being destroyed, or because another activity (either an existing one or a new one) has been resumed and is covering it.
	Followed either by `onRestart()` if the activity is coming back to interact with the user, or by `onDestroy()` if this activity is going away.
		Yes	`onRestart()`
			or
			`onDestroy()`

`onDestroy()`	Called before the activity is destroyed. This is the final call that the activity will receive. It could be called either because the activity is finishing (someone called `finish()` on it), or because the system is temporarily destroying this instance of the activity to save space. You can distinguish between these two scenarios with the `isFinishing()` method.	**Yes**	*nothing*

4.3 Code Analysis

The activities of our application are the following two:

1. ***SimpleCalendarViewActivity***
2. ***TestingMapsActivity***

4.3.1 *SimpleCalendarViewActivity[9]*

4.3.1.1 General Description

This is the primary activity of our application and it is used for the application's launch. It contains methods that are used to populate the initial screen with image objects, button assignments that make the transitions to the user's calendar or the map Activity and algorithms that display pop up messages based on the network status and the calendar events.

4.3.1.2 Class *SimpleCalendarViewActivity*

4.3.1.2.1 Variables

In this section we give a short description of the global variables used in this activity.

Variable	Description
input	This variable is used to store the event's locations. It is used as static because it will be transferred to the mapActivity
help_input	The purpose of this variable is the same as above only in this case we use it when the user decides to change the order of event's
Recalculation	This Boolean is used to check if the user decides to recalculate the optimal route. If so then the previous countdown timer is stopped and a new one is initiated.

Variable	Description
selectedDayMonthYear	This button is used to display the selected day, month, year
ViewMap	This button is used to initiate the mapActivity
currentMonth_button	This button displays the current Month.
viewData	This button is used to initiate the agenda activity to display the events of a selected date.
prevMonth	Button that makes the switch to the previous month
nextMonth	Button that makes the switch to the next month
CalendarView	This variable is used to display the calendar screen in a gridview style as it is mentioned before
Month, year	Integers with the number of month and year
enter_data	Button used to initiate the phone's calendar activity to schedule an event on a selected date
temp2	In this string we store the selected date, to display it to the selectedDayMonthYear button.
dateTemplate	String that shows the format of date
temp3	In this array we store the days, months and years numbers separately as a result of the split method applied to *temp2*
esoteriko	This Boolean is used to distinguish if the map activity has been called from the initial screen or from inside the map.

Variable	Description
cal	It is a GregorianCalendar variable that will be used to construct a new GregorianCalendar initialized to midnight in the default TimeZone and Locale on the specified date.
helpCurrentDay	In this variable we store the days' number that we acquire from the first position of array *temp3*
helpCurrentMonth	In this variable we store the months' number that we acquire from the second position of array *temp3*
helpCurrentYear	In this variable we store the years' number that we acquire from the third position of array *temp3*
Locations	In this ArrayList we store the events' locations of one selected date
help_locations	This ArrayList has the same purpose as mentioned above but is used in case the user decides to change the order of events.
descriptions	In this ArrayList we store the events' descriptions on a selected date
Help_descriptions	This ArrayList has the same purpose as mentioned above but is used in case the user decides to change the order of events.
times	In this variable we store the events' start time.

4.3.1.2.2 Methods

In this section we present only the methods that were not discussed in Section 4.2.5.

- **private void** setGridCellAdapterToDate (**int** month, **int** year)

  An Adapter object acts as a bridge between an AdapterView and the underlying data for that view. The Adapter provides access to the data items. The Adapter is also responsible for making a View for each item in the data set. In this method we use the adapter to set each grid on the calendar launch screen to display the correct day, month, and year.

- **public void** onClick(View v)[18,19,22]

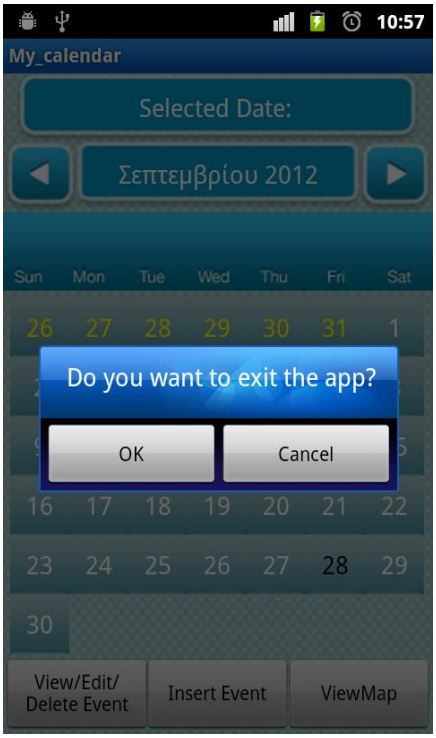
  This method is called when a ViewObject has been clicked. In this method we check the following five cases:

  1. User clicks the prevMonth button whick means the app has to display the days of the previous month.
  2. User clicks the nextMonth button which means the app has to display the days of the next month
  3. User clicks the enter_data button which means the user wants to add an event. When this action is chosen the following procedure is initiated:
     (a) Check if the user has selected a date. If not, a pop up message will notify him to do so.
     (b) Read the current date and time from the phone's clock and open the calendar to this specific date. Also we set the end time of the event one hour after the starting time.
  4. User clicks the viewData button which means he wants to see the events of a selected date. In this case the following procedure is initiated:
     (a) Check if the user has selected a date. If not, a pop up message will notify him to do so.
     (b) Read the current date and time from the phone's clock and open the agenda to the specific date.

5. User clicks the viewMap button which means he wants to see the optimal route of the calendar's events on a selected date. In this case the following procedure is initiated:
   (a) Check if the user has selected a date, if not then a pop up message will notify him to do so.
   (b) Check if the data network is active (wi-fi, 3G), if not a pop up message will notify him to do so. It is mandatory for the app to work an active internet connection.
   (c) I read the calendar's events on a selected date, and store in variables the event's locations, descriptions and starting time.
   (d) I check if the *input* variable which contains the locations is empty. If so, then a pop up message will notify the user about this. If not, then the mapActivity will initiate to display the optimal route based on the data acquired.

- **public void** onBackPressed()

Called when the activity has detected that the user pressed the back key. The default implementation simply finishes the current activity but in this case we override it to display an alert dialog which asks the user whether he wants to exit the app or not. There are two possible options here, **OK** and **Cancel**. If the user clicks on OK the app exits, or if he clicks Cancel the app returns to the initial screen.



4.3.1.3 class GridCellAdapter

This inner class in **SimpleCalendarViewActivity** is used to tailor the basic adapter that android provides us to our needs. The code was taken from this source, and builds a simple calendar view with the right transition between months, years and days.[3]

4.3.1.3.1 *Variables*

Variable	Description
_context	Interface to global information about an application environment. This is an abstract class whose implementation is provided by the Android system. It allows access to application-specific resources and classes, as well as up-calls for application-level operations such as launching activities, broadcasting and receiving intents, etc.
list	A List is a collection which maintains an ordering for its elements. Every element in the List has an index. Each element can thus be accessed by its index. In this list we add the month days.
Months	In this array we keep all the months' names.
daysOfMonth	In this array we keep the number of days that each month has.
daysInMonth	In this integer we store the number of days per month.

Variable	Description
currentDayOfMonth	In this integer we store the number of this month's current day
currentWeekDay	In this integer we store the number of the current week's day
gridcell	we use this variable to display data on the launch screen

4.3.1.3.2 Methods

- **private void** setCurrentDayOfMonth(**int** currentDayOfMonth)
  This method is used to set the current day of the month.

- **public void** setCurrentWeekDay(**int** currentWeekDay)
  This method is used to set the current week day.

- **private void** printMonth(**int** mm, **int** yy)
  This method is used to add to the list variables mentioned before the correct dates.

- **public** View getView(**int** position, View convertView, ViewGroup parent)
  This method is used to display the data at the specified position in the data set
  Parameters

  position The position of the item within the adapter's data set of the item whose view we want.

  convertView The old view to reuse, if possible.

  parent The parent that this view will eventually be attached to

Returns A View corresponding to the data at the specified position.

- **public void** onClick(View view)
  This method is called when a ViewObject has been clicked. In this method we do the following:
  (1) Assign in the selectedDayMonthYear button the selected day so it can be displayed in the initial screen
  (2) Read the selected date and split it in three parts (day, month, year)
  (3) Read all events of the selected date and display a pop up message to inform the user about their number.

- **private** String getMonthAsString(**int** i)
  With we can retrieve a month's name from the *months array based on the month's number.*

- **private int** getNumberOfDaysOfMonth(**int** i)
  With this method we can retrieve the total number of a month's days based on the month's number.

- **public** int getCurrentDayOfMonth()
  This method is used to get the number of the current day.

4.3.2 TestingMapsActivity

4.3.2.1 General Description

This activity is launched when the user clicks the viewMap Button.[4.3.1.2.1]. It displays the map that contains the optimal route. Moreover in this activity there are algorithms that

1) calculate the optimal route,
2) track user's current position every 15 seconds,
3) change the order of destinations according to user's wishes,
4) toggle between satellite and street view,
5) recalculate optimal route if an error occurs,
6) view an event at a selected location on the map,
7) display the address of a selected location of the map
8) check if there is an active GPS connection, and if there isn't any an alert dialog prompts the user to activate the phone's GPS
9) Alert the user to start in 10, 5, 0 minutes in order to be at his appointment at the scheduled time.

4.3.2.2 class TestingMapsActivity

4.3.2.2.1 Variables

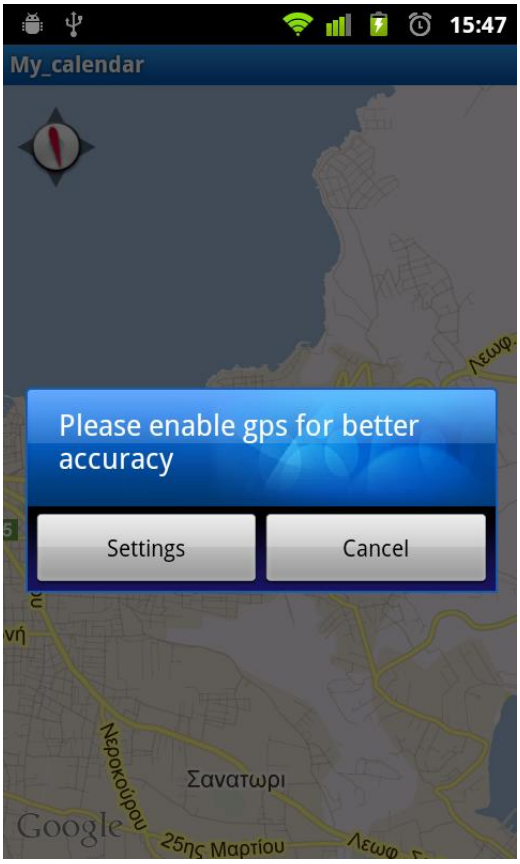
Variable	Description
overlaylist	This type of list is used to display items on the map
mapView	Our map
touchedPoint	This variable is the position that the user touched on the map
pointToDraw	This variable has the coordinates of a map's location that belongs in the optimal route
Lm	With this variable we use the location services available by google
obj	This variable is used to call the permutation algorithm
check	In this variable we store the provider's name (GPS, Wi-Fi).
timeToDestination	In this integer we store the estimated time arrival that google servers return between two locations
Start, Stop	These two variables are used to count the time from the moment the user presses the screen to the moment he removes his finger.
time	This variable contains the total time to visit a route.
Geopoints	This list contains all the places in the database with the form of coordinates.
array	This array of Booleans is used to mark each location as visited or not.

Variable	Description
Check2	This Boolean is used to check if the location that the user touched on the map exists on the database
GetMemonomenhDiadromh	This variable is used to display the locations in order for the user to change their order.
Stop_Timer	This Boolean is used to stop the countdown timer.
pause	This Boolean is used to pause the app if it is not in the foreground.
thesh	This variable is used to store the location's position in the list which the user has selected to visit first.
lo	This variable is used to track our current position and activate compass function on the map
Message1	Check if the message for the 10 minute warning has been shown.
Message2	Check if the message for the 5 minute warning has been shown.
Message3	Check if the message for the 0 minute warning has been shown.
dialog	A progress dialog that informs the user that the map is loading.

4.3.2.2.2 Methods

- **private void** gpsMessage()

This function checks if the user has activated the phone's GPS. If not, then an alert dialog prompts the user to do so. If the user agrees to activate the GPS, then by clicking the settings button he is transferred to the phone's settings. If he clicks Cancel then he returns to the map screen. In case the gps is active before the activity starts then nothing is shown.



- **private void** timer2()

This function is used to start the countdown time that delays the main algorithm to start by 2 seconds in order to give time to the map to load.

- **private void** teliko(String[] ksexwristes_diadromes)[24,25]

This function contains the major part of our algorithm that computes the optimal route and draws it on the map.

- **private** void timer()

Countdown timer used to call teliko and teliko2 functions every 15 seconds.

- **private** String[] vres_diadromes(Collection<String> input)
  This function calls the permutation algorithm used to calculate all the possible routes.

- **private int** calculation(GeoPoint src, GeoPoint dest)[15]
  In this function we calculate the estimated arrival time from one geopoint to another.

- **private void** DrawPath(GeoPoint src, GeoPoint dest, **int** color, MapView mMapView) [24.25,38]
  This function has the code that establishes the connection with Google's servers to get the location's coordinates.

- **private** List<GeoPoint> decodePoly(String encoded)[24.25,38]
  This function decodes the encoded message that comes as a response to our request from the servers.

- **private** String makeUrl(GeoPoint src, GeoPoint dest)[24,25,38]
  This function is used to remake our request for the servers in a form that they will understand.

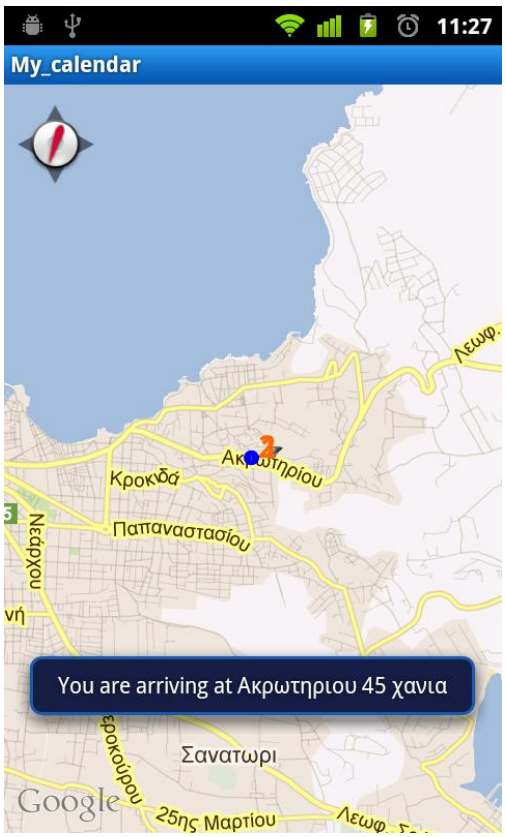
- **private** Drawable createFromView(**int** positionNumber)[4]
  This function is used to draw a number on each location on the optimal route.

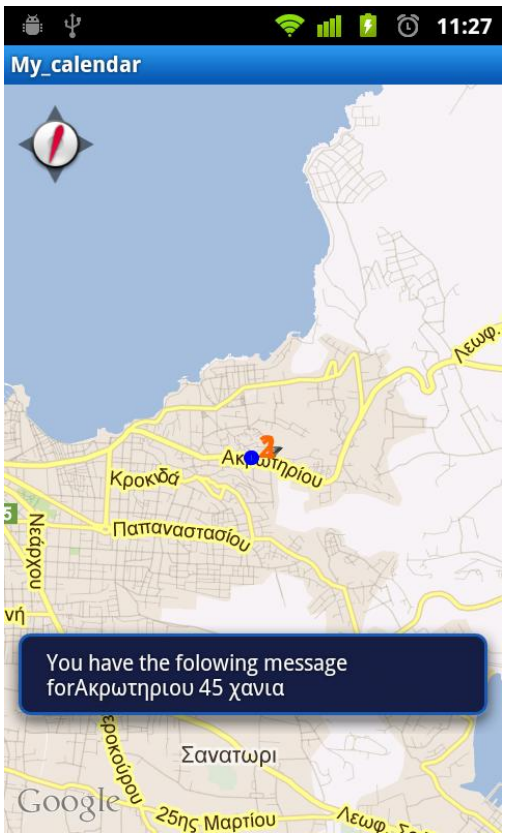
- **protected void** onPause()
  Function that pauses the activity when the app goes to background and disables the compass function

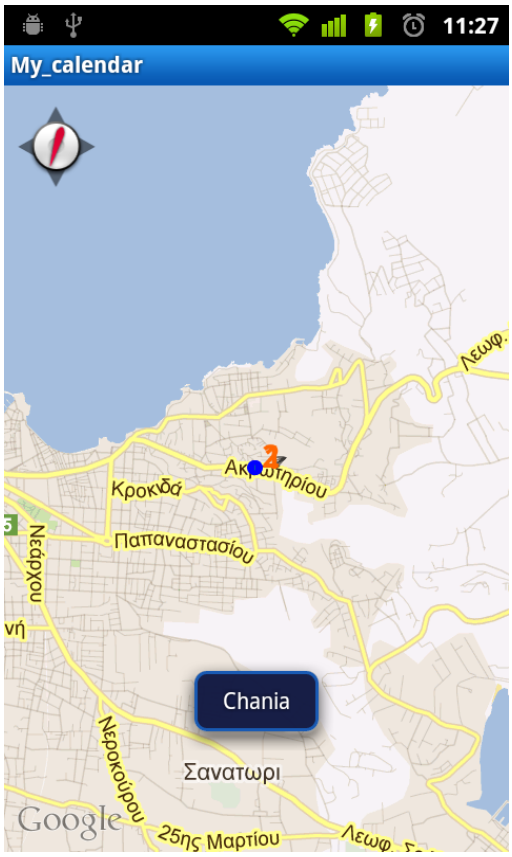
- **protected** void onResume()
  Function that resumes the activity when the app comes again on the foreground.

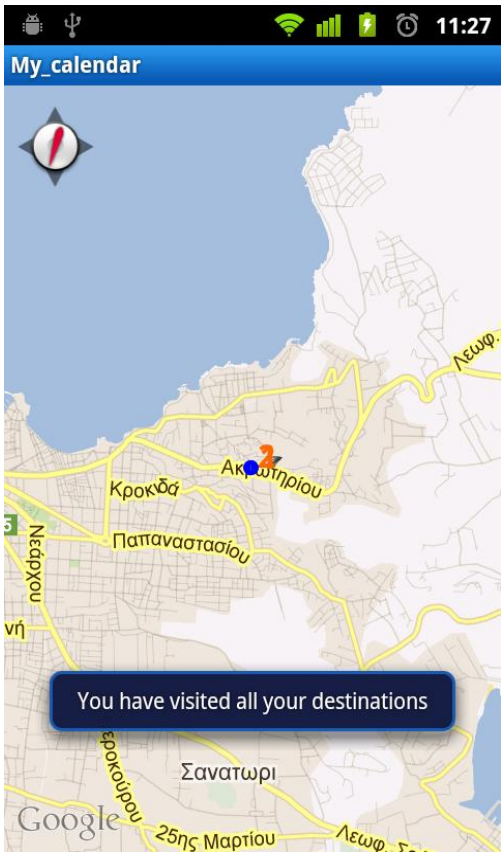
- **protected** boolean isRouteDisplayed()
  Function that is used for accounting purposes. The server needs to know whether or not any kind of route information, such as a set of driving directions, is currently being displayed.

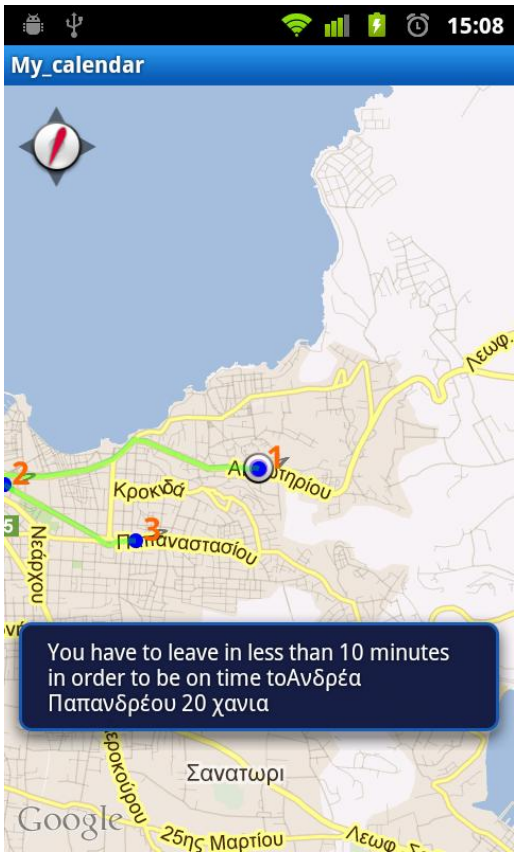
- **private** void check_location()
  This function does all the background checking such us:
  - tracking down the user's position every 15 seconds
  - informing the user about his arrival at a destination point, when he at a radius of 120 meters around this point.
  - informing the user about his the route's end when he has visited all the locations,
  - informing the user about location empty events
  - informing the user about messages in this location
  - Finally informing the user with warning messages 10,5,0 minutes before his immediate departure in order to keep the schedule.

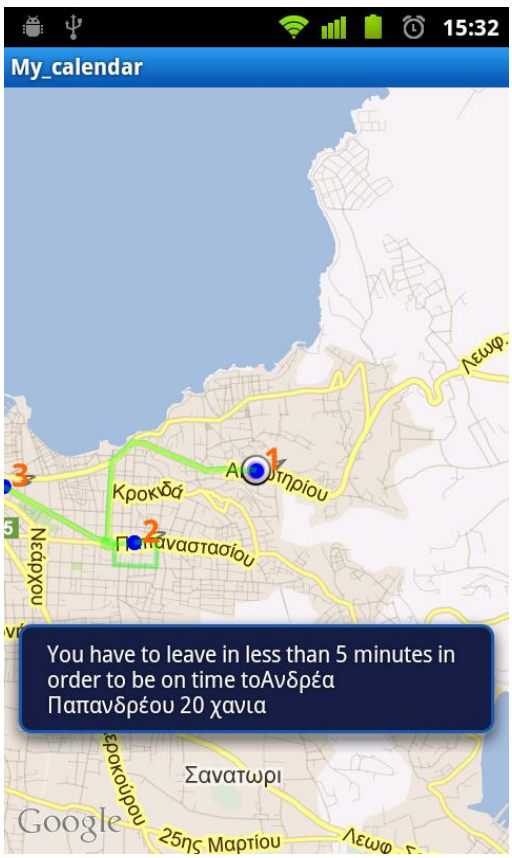





11:27
My_calendar
Κροκιδά
Παπαναστασίου
Σανατωρι
Chania
Google
25ης Μαρτίου


11:27
My_calendar
Κροκιδά
Παπαναστασίου
Σανατωρι
You have visited all your destinations
Google
25ης Μαρτίου


15:08
My_calendar
Κροκιδά
Σανατωρι
You have to leave in less than 10 minutes in order to be on time toΑνδρέα Παπανδρέου 20 χανια
Google
25ης Μαρτίου


15:32
My_calendar
Κροκιδά
Σανατωρι
You have to leave in less than 5 minutes in order to be on time toΑνδρέα Παπανδρέου 20 χανια
Google
25ης Μαρτίου

- **private void** teliko2(String[] ksexwristes_diadrome)

  This function does exactly the same as ***teliko*** but it is called only when the user decides to change the order of locations of the route.

- **public void** onBackPressed()

  Called when the activity has detected that the user pressed the back key. The default implementation simply finishes the current activity but in this case we override it to terminate the current process and free all resources allocated.

4.3.2.3 class MyOverLay

This inner class is called from the DrawPath( ) method to draw the route on the map.

4.3.2.3.1 Variables

Variable	Description
pathColor	This variable is the path color, green in our case
points	The list of geopoints that have been downloaded from the server.

4.3.2.3.2 Methods

- **public Boolean** draw(Canvas canvas, MapView mapView, **boolean** shadow,**long** when)

  This function sets the starting and ending point in the route and then calls the drawOval( ) method to draw the route between them.

- **private void** drawOval(Canvas canvas, Paint paint, Point point)

  This method draws the route between two geopoints on the map.

4.3.2.4 class gia<T>

This class contains only one method which is the permutation algorithm to find all possible routes.

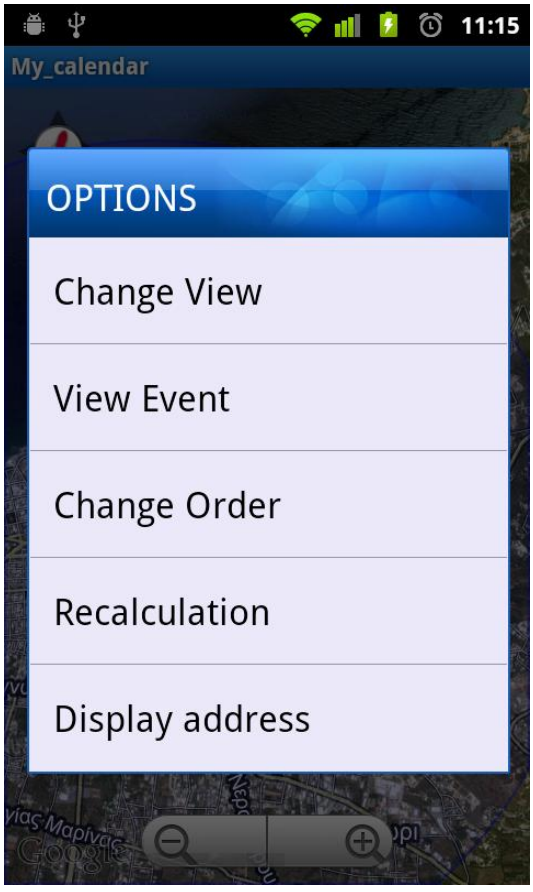
4.3.2.4.1 Methods

- **private** Collection<List<T>> permute(Collection<T> input)
  This is the permutation algorithm that generates all possible routes. It will be explained in detail in the next section.

4.3.2.5 class TouchOptions

As the name suggests this class contains an option menu that is integrated into the map. In order to view this menu the user must touch the map for over 1,25 seconds. The menu's options are:

1. Change between satellite and street mode
2. View the message of a touched location (if there is any).
3. Change the order of destinations by giving priority to one of them.
4. Recalculate the route if it is not displayed correctly or to view the original route if you have previously chosen the second option.
5. Display the address of a touched point on the map.



4.3.2.6 Public class PinpointClass

This class, although it is not an inner class is used in the **TestingMapsActivity class** widely. With this class we insert a pinpoint on the map such us the numbers on the locations. It needs 2 assignments:

i) The drawable item, in this case the location's order number
ii) The context of the application which gives us access to Map View.

4.3.2.6.1 Variables

Variable	Description
pinpoints	The list of pinpoints
ctx	The app's context

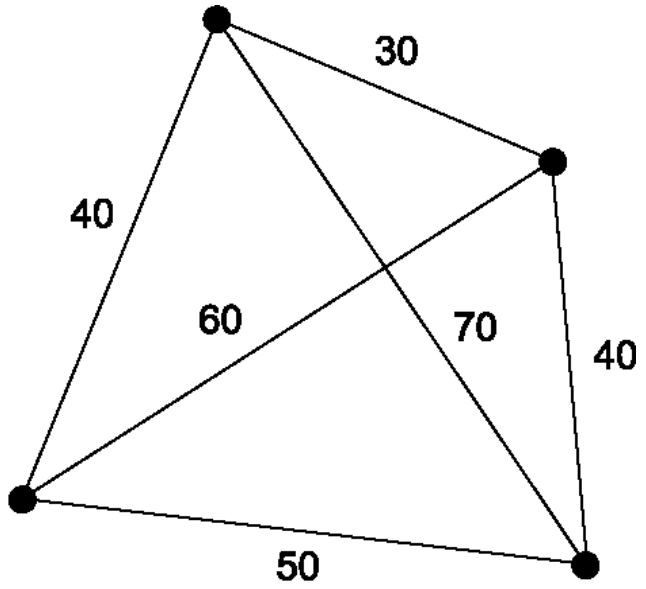
4.3.2.6.2 Methods

- **public void** InsertPinpoint (OverlayItem item)
  This method adds the pinpoint to the pinpoint's list in order to display it on the map.

Chapter 5: ALGORITHM ANALYSIS

5.1 Graph Problem

Our problem can be modeled as an undirected weighted graph, where locations are the graph's vertices, paths are the graph's edges, and a path's distance is the time between two vertices. It is a minimization problem starting at a specified vertex and ending after having visited each vertex exactly once. It is very similar to the travelling salesman problem (TSP) with the only difference that we do not return to the starting location when we have visited all our destinations. It can be easily though transformed into a TSP problem by adding the starting location as a calendar event and then choosing to change the order of the events. The following graph, containing random values, gives a generic representation of the problem.



5.2 Possible solutions

- **Prim's algorithm**
- **Kruskal's algorithm**
- **Nearest Neighbor algorithm**
- **Dijkstra's Algorithm**
- **Permutation algorithm**

In order to pick the right algorithm in order to find the optimal route we followed the method of elimination.

- Prim's and Kruskal's algorithms were the first to eliminate. These two algorithms are used to find minimum spanning trees (MSTs), i.e., a tree that connects all vertices. Degree-constrained MSTs are NP-hard problems. In fact, for any fixed constant $d \geq 2$, the problem of finding a minimum cost degree-constrained spanning tree of degree at most d is still NP-hard [11-13]. Also, for any fixed rational $\alpha > 1$ and any fixed d, finding a spanning tree with degree constraint d and cost within a factor α of the optimal is NP-hard. Various heuristics solutions have been proposed (see [11] and references therein), which work well but do not produce the optimal route, which is our goal in this thesis.
- Dijkstra's Algorithm was the next one to be eliminated. This algorithm is a single source shortest path algorithm that works well when the goal is to find the path between two points on the map. Our problem is to define the shortest

route between all the designated map points, for which Dijkstras algorithm does not provide an optimal solution [17].

- The last algorithm eliminated is the Nearest Neighbor (NN) algorithm. Although this algorithm appears to be a logical solution since every path is the shortest path between two nodes, it fails greatly to deliver the optimal path. The reason is that NN tries to find the local optimum, in each case, whereas, we are looking for the global optimal route. For N cities randomly distributed on a map the algorithm on average yields a path 25% longer than the shortest possible path. Also, there exist many specially arranged city distributions which make the NN algorithm give the worst route [21].

Therefore, we used the exhaustive permutation algorithm because:

1) It guarantees that the solution will be the global optimum.
2) An average mobile user is not likely to have to visit over 5 places in one day and even if he does, the permutations generated don't need much CPU time for today's CPU's. The computational part of our measures needs less than 5 seconds even for 10 entries.

5.3 Permutation Algorithm

5.3.1 Analysis

```
private Collection<List<T>> permute(Collection<T> input) {
                Collection<List<T>> output = new
ArrayList<List<T>>();
                if (input.isEmpty()) {
                      output.add(new ArrayList<T>());
                      return output;
                }
                List<T> list = new ArrayList<T>(input);
                T head = list.get(0);
                List<T> rest = list.subList(1, list.size());
                for (List<T> permutations : permute(rest)) {
                      for (int i = 0; i <= permutations.size();
i++) {
                            List<T> subList = new
ArrayList<T>();
                            subList.addAll(permutations);
                            subList.add(i, head);
                            output.add(subList);
                      }           }
                return output;   }
```

The algorithm accepts one argument which is the list of locations. If the list is Empty then it returns an empty list because no permutations can be generated.

If the list is not empty, then for every permutation we call the method recursively. In this way we create sublists that contain all the original locations but in every different order. We add these to the output list which is essentially a large list in which each element is a sublist that contains a permutation of the original locations.

```
ksexwristes_diadromes = pnr.toString().split("],");
```

After retrieving the output, we split it twice. The first time happens in order to convert the list into an array(ksexwristes_diadromes) where each element contains one possible route.

```
for (j = 0; j < ksexwristes_diadromes.length; j = j + 1) {
                int time = 0;
                memonomenes_diadromes =
ksexwristes_diadromes[j].split("\\,");

                for (int i = 0; i <
memonomenes_diadromes.length; i = i + 1) {
                     dest_lat = 0;
                     dest_long = 0;
                     Address loc = null;

                     try {
                          memonomenes_diadromes2 =
memonomenes_diadromes[i];
                          address_dest =
coder.getFromLocationName(
                                    memonomenes_diadromes2,
1);

                          if (address_dest.size() > 0) {
                               loc = address_dest.get(0);
                               dest_lat = loc.getLatitude();
                               dest_long =
loc.getLongitude();
                          }
                     } catch (IOException e) {
                          e.printStackTrace();
                     }
          Geopoints.add(i + 1, (new GeoPoint((int) (dest_lat *
1E6),
     (int) (dest_long * 1E6))));

     time = time     + calculation(Geopoints.get(i),
Geopoints.get (i + 1));
     }
time2.add(j, time);}
```

Then we split this array into multiple arrays, each one containing one possible route. In these new arrays every position corresponds to a location. We store this location to a variable (memonomenes_diadromes) and request the ETA from our current position to this location. In the next iteration we request the ETA from this location to the next one and so on until we find the total ETA time from our current position to the last location of this route. This procedure is repeated for every possible route in order to find the minimum ETA that corresponds to the optimal route.

```
Geopoints.add(i + 1, (new GeoPoint((int) (dest_lat *
1E6),(int) (dest_long      * 1E6))));

 DrawPath(Geopoints.get(i), Geopoints.get(i + 1),
Color.GREEN,mapView);
```

When we find this route we add each location's Geopoint to our *Geopoints* list and draw the route for these Geopoints.

5.3.2 Complexity and Proof

In order to execute the permutations we had to define some criteria.

1) Repetitions are not allowed.
2) Order is important.

Based on this criteria we know that the number of permutations can be extracted from the following mathematic formula:

$$P(n,r) = \frac{n!}{(n-r)!}$$

n=number of locations
r=locations chosen

- **Permutation formula proof**
  - There are *n* ways to choose the first element
    - *n*-1 ways to choose the second
    - *n*-2 ways to choose the third
    - …
    - *n*-*r*+1 ways to choose the r^{th} element
  - By the product rule, that gives us:

  $P(n,r) = n(n-1)(n-2)\ldots(n-r+1)$

  $P(n,r) = n!/(n-r)!$

Hence this exhaustive algorithm has a complexity of O (n!), which means that for a large number of locations (n>100) it would be difficult to generate the permutations in a tolerable amount of time. Actually, it needs to be emphasized that the permutation algorithm is the most computationally complex, in comparison to all the algorithms

discussed in Section 5.2. A computationally simpler solution would be to retain in memory every ETA, in order not to have to request it again for the next permutation when we will need it. Due to the possibility of often changing traffic conditions, this approach might not always provide the optimal route but is of a much lower complexity. This approach will be used in future versions of our application, in order to compare its results and efficiency (in terms of delays) against the exhaustive permutation algorithm.

5.3.3 Experimental Times

During our application testing, we measured the following average completion times, depending on the number of locations.

# Locations	Completion Time*
0	1 second
1	<3 seconds
2	<7 seconds
3	<17 seconds
4	<55 seconds
5	Almost 5 minutes
6	Not recommended

These observations are based on Wi-Fi connections. For 3G connections, we encountered 50% - 70% additional delay, due to the smaller connection speeds. We define Completion time as:

*Completion time= Map Loading Time + Route Drawing Time + Received Data Time + Permutation Algorithm Execution

Hence, completion time is the sum of four different times. However, each of these times holds a different weight in this sum. Basically this rule applies:

Received Data Time >> Permutation Algorithm Execution > Map Loading Time = Route Drawing Time

The first 2 times have a minimal impact on the completion time. When the input has no locations, **Completion Time** is calculated by the sum of these two arguments since the other two are zero and is equal to 1 second as seen on the table above.

The **Received Data** and **Permutation Algorithm Execution** times are the ones that contribute the biggest part on the total delay. The delay increases exponentially for every location increment by 1 due to the O (n!) complexity of the permutation algorithm. Although the CPU time required is small even for 10 entries, the big problem lies in the large number of permutations generated. For each permutation we must calculate the total time to complete the route and to do that we must send requests and receive responses from the servers.

To give a simple example consider the following case:

- We have 3 locations to visit in a certain day
- We must find the permutation of these cities by combining them into groups of three.
- This will generate P (3,3) = 6 permutations.
- Each permutation contains 3 elements which are the addresses.
- We must find the ETA between current location and these addresses in the way described in [5.4.1].
- This will give us 3 pairs of requests and responses per permutation.
- So we have a total of 18 pairs of requests and responses for all permutations generated.
- Considering the network lag for every pair of requests and responses we can see that for a large number of entries the **Received Data Time** is large and can have a serious impact on the **Completion Time.**
- For an input of 5 cities we have P (5,5) =120 permutations and a total of 600 pairs of requests and responses!

5.4 Other Algorithms

Besides the main algorithm that we analyzed in the previous section, there are other algorithms that run in the background and give the app navigation functionality. These algorithms are used in **check_location** method and the **TouchOptions** class (sections 4.3.2.2.2 and 4.3.2.4 respectively).

5.4.1 Algorithm in check_location

This algorithm reads the location database every 15 seconds and compares the current location with all the locations in database. We compare the first five digits of the address's geopoint with the 5 digits of the current location's geopoint, thus creating a radius around each location of about 120 meters. When a user enters the radius, triggers are activated and pop up messages appear with the location's name and message.

```
if ((help_lat3[0] == help_lat[0])&& (help_lat3[1] ==
help_lat[1])&& (help_lat3[2] == help_lat[2])
&& (help_lat3[3] == help_lat[3])
&& (help_lat3[4] == help_lat[4])
&& (help_lon3[0] == help_lon[0])
&& (help_lon3[1] == help_lon[1])
&& (help_lon3[2] == help_lon[2])
&& (help_lon3[3] == help_lon[3])
&& (help_lon3[4] == help_lon[4])
&& array[i] != true) {
      Toast.makeText(this,"You are arriving at "+
SimpleCalendarViewActivity.locations.get(i),
Toast.LENGTH_LONG).show();
if
(!SimpleCalendarViewActivity.descriptions.get(i).isEmpty())) {
Toast.makeText(this,"You have the folowing message for"+
SimpleCalendarViewActivity.locations.get(i),
Toast.LENGTH_SHORT).show();
Toast.makeText(this,SimpleCalendarViewActivity.descriptions.ge
t(i), Toast.LENGTH_LONG).show();
                                    }
                                    array[i] = true;
                                                }
```

Additionally we check if we have visited all destinations or if there is a problem with a specific location (such us no geopoints applied to it).

5.4.2 Algorithms in TouchOptions

In this class there are two algorithms worth mentioning.

1) The first algorithm is to View the message at a touched location. This algorithm uses the exact same methodology as in section [5.4.1].
2) The second one is the one that changes the location's order on the route. Its function is as follows:
   - The geopoint of the user's selected location is being inserted to the second position of the Geopoint's list.
   - Hence the first two positions are occupied by the current location and the user's selected location.
   - For the next spots the permutation algorithm is applied, repeating the procedure from the start.

The complexity of these algorithms is O (n), since only a simple reading on the lists is required and some comparisons.

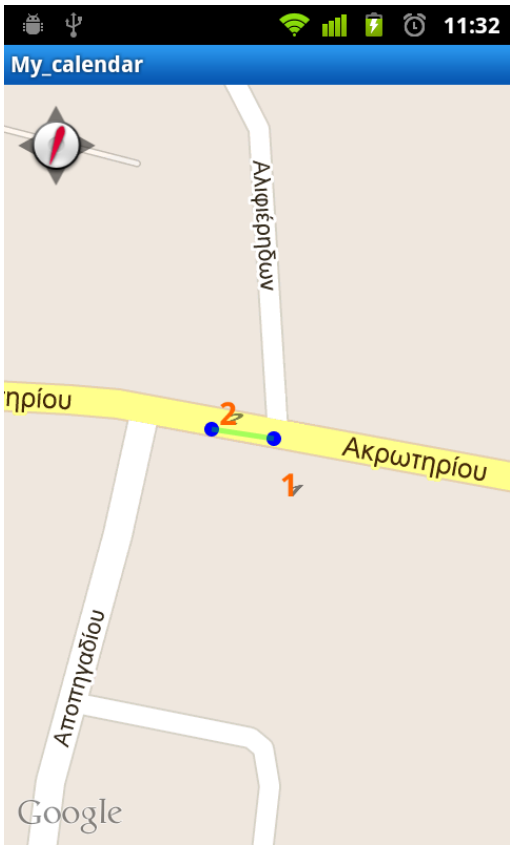
Chapter 6: FUNCTIONALITY ISSUES

6.1 Acquiring Current Location

In order for the app to work properly, the acquirement of current location must be precise. For this reason the user is notified immediately if the phone's GPS is turned off.

In the case that GPS is not available when the localization starts then the results might not be accurate. If this occurs, there are two cases:

1) There is a Wi-Fi network available, so the app uses this network-type to acquire the user's location. In this case there can be an offset of 50 meters but the real problem occurs while the user is out of range. The app's behavior can be unpredictable in this case, it can either display a warning message or even crash.



In the picture above we can see that for the same address 2 different locations have been acquired. The first location is acquired from Wi-Fi whereas the second one from the server. As we can see there is an offset of a few meters (42 meters) between them. The second one is the most accurate since it is being retrieved from Google satellites database, but in the end this small offset will not have any serious impact on the app's functionality.

2) There is only 3G availability. In this case the app's functionality is limited. It is almost impossible to acquire user's location using cell towers. The offset can be from a few hundred meters to some kilometers depending on the tower's location. Furthermore, at times when the server cannot find via 3G the user's location it just sends the last known location from its database. In either case the optimal route is completely wrong since the starting point is wrong.

NETWORK TYPE	OFFSET(meters)	ACCEPTABLE
Wi-Fi	50	Yes
3G	>1000	No

6.2 Update Interval

As mentioned in Section [4.3.2.1] the update interval is 15 seconds. If for some reason there is no data connection available (via 3G or Wi-Fi) and during this time no data will be received, then a second request will be sent while the first is still incomplete. In this case the app's behavior is unpredictable. It can either display warning messages about the locations or can even crash.

6.3 Screen Freezing

During the update procedure the screen might temporarily freeze. This happens because the app is trying to run all background algorithms and redraw the map. The freeze lasts for a fraction of second and might not even happen in smartphones with up to date hardware. In case such a freeze occurs, Android thinks that the app is going to crash and displays a message that asks the user

- either to force-terminate the app
- or wait

In this case the **wait** option must be selected; then the screen unfreezes immediately.

6.4 All day Events

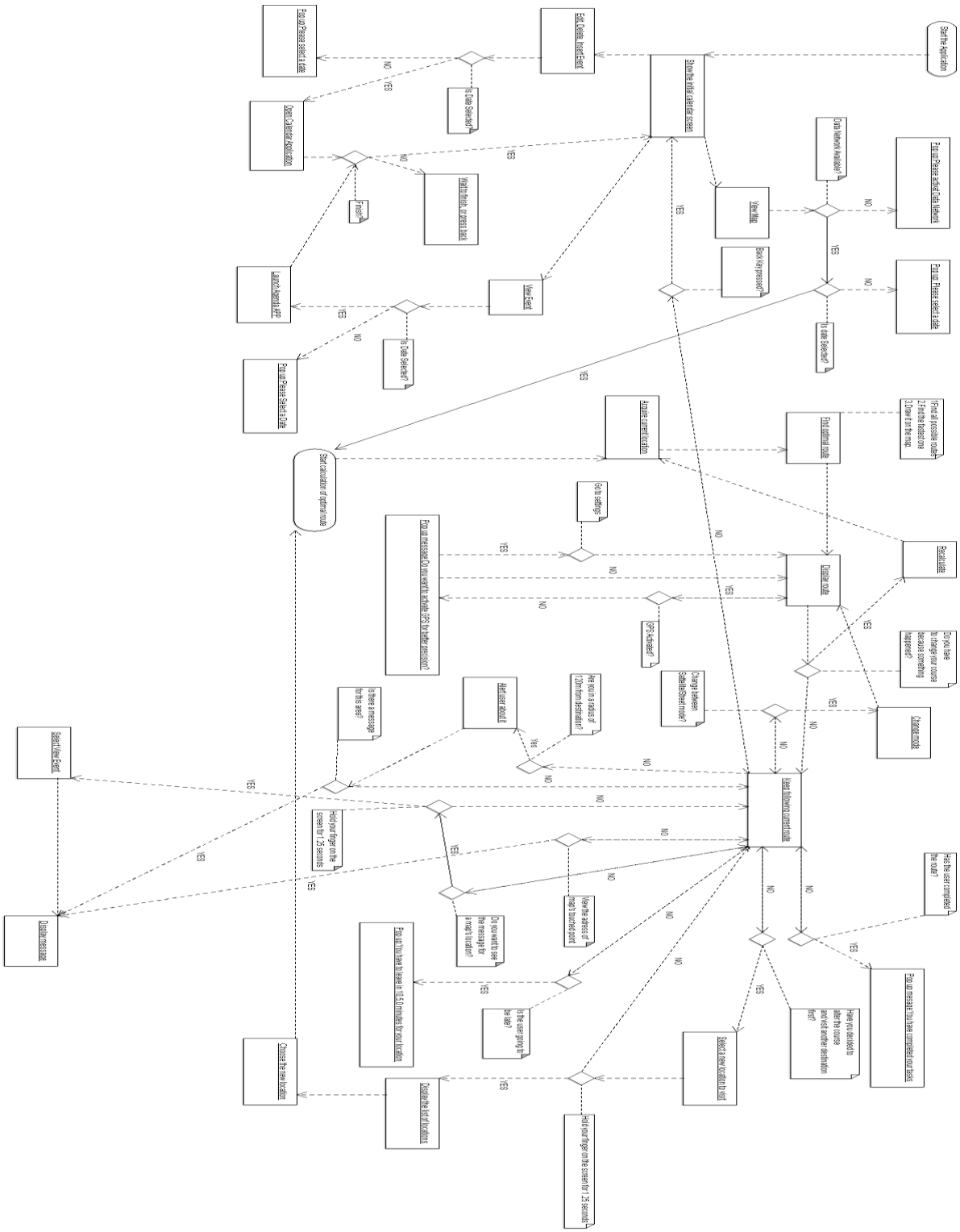
Google calculates time in each region based on Greenwich Mean Time (GMT). Therefore, all day events differ from region to region. In Greece there is a time difference of +3 hours (GMT+3:00) from Greenwich, thus an all day event is considered to last from 3a.m of current day to 3a.m of the next day. This creates a problem when calculating the optimal route of the daily events because the algorithm adds the last day's all day event to the location's database. As a result we have an increased complexity and a wrong optimal route.

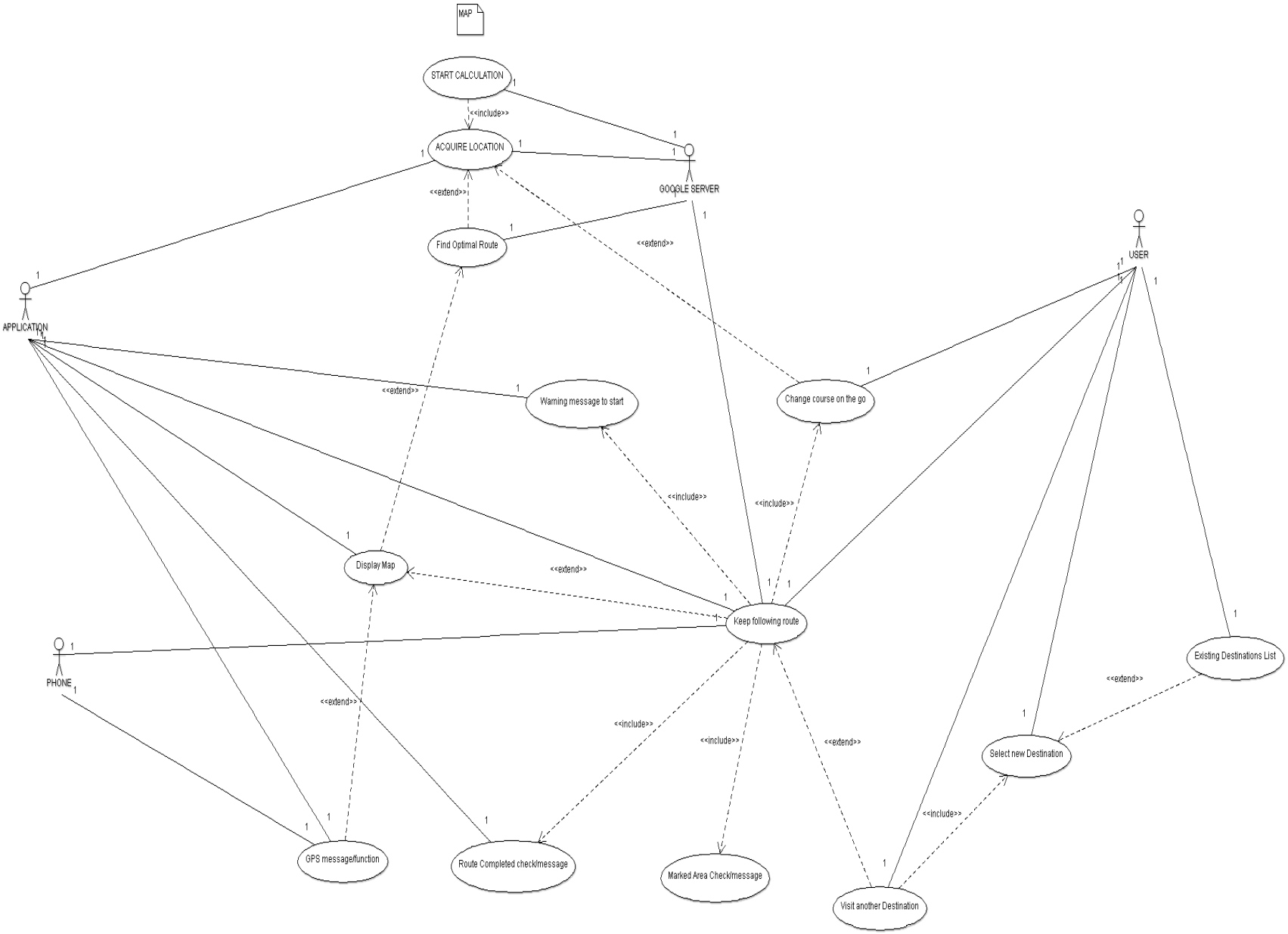
6.5 Calendar Applications

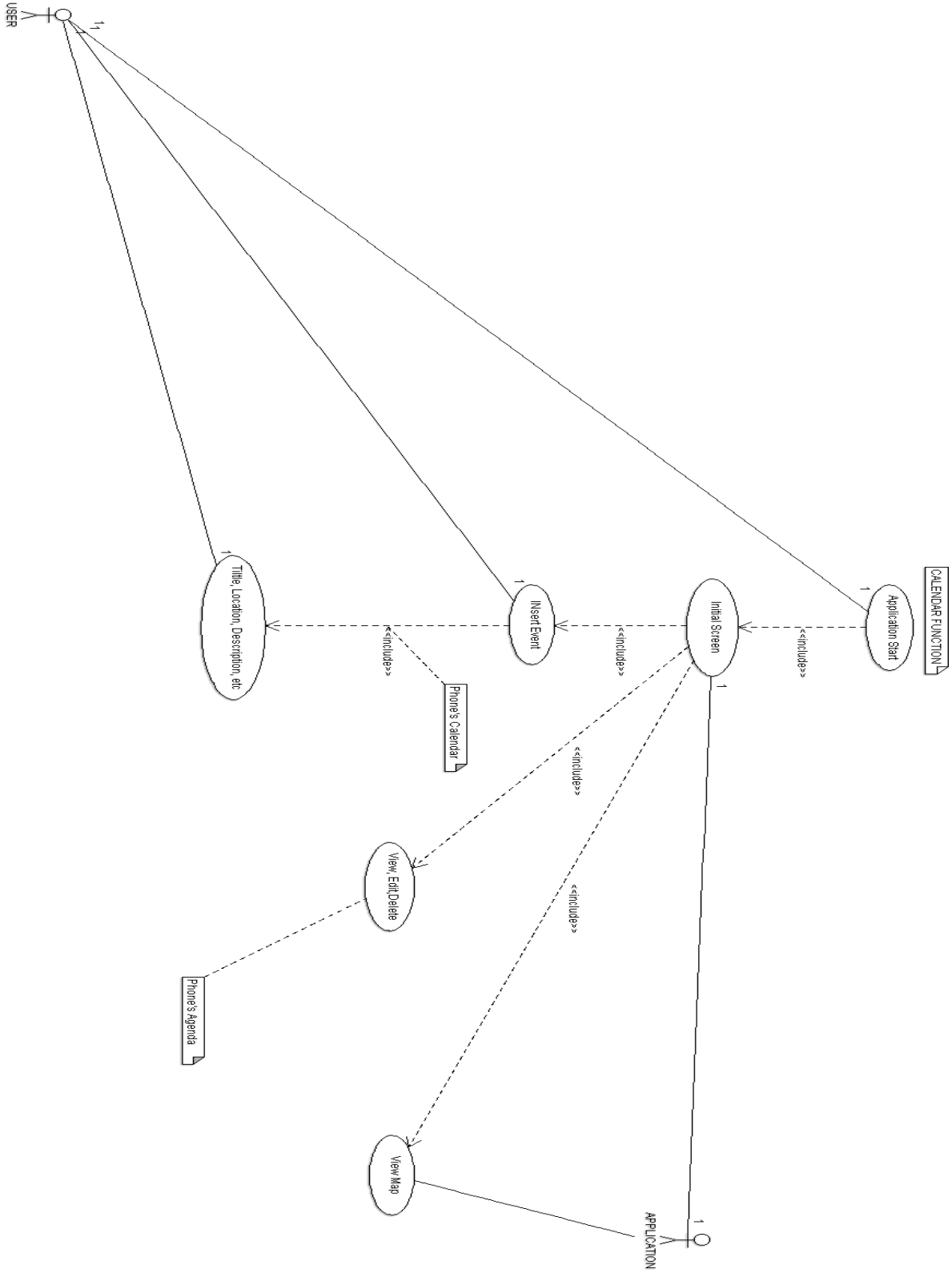
For the time being this version of the app works only with the default calendar application. The synchronization with the online Google calendar is not yet supported.

Chapter 7: Requirements Analysis

7.1 Final Flowchart Diagram,Use case Diagrams




MAP
START CALCULATION
<<include>>
ACQUIRE LOCATION
GOOGLE SERVER
<<extend>>
Find Optimal Route
<<extend>>
USER
APPLICATION
<<extend>>
Warning message to start
Change course on the go
<<include>>
<<include>>
Display Map
<<extend>>
Keep following route
PHONE
Existing Destinations List
<<extend>>
<<extend>>
<<include>>
<<include>>
<<extend>>
Select new Destination
<<include>>
GPS message/function
Route Completed check/message
Marked Area Check/message
Visit another Destination


USER
1
1
CALENDAR FUNCTION
Application Start
1
<<include>>
Initial Screen
1
<<include>>
INsert Event
1
<<include>>
Tiitle, Location, Description, etc
Phone's Calendar
<<include>>
View, Edit,Delete
Phone's Agenda
View Map
APPLICATION
1

7.2 Use Case Diagram analysis

Use Case 1	**Start the Application**	
Goal In Context	Successful start of our application	
Preconditions	User has found the app's icon from the his phone's app drawer	
Success End Condition	User has successfully launched the app and sees the initial screen	
Failed End Condition	App didn't launch and the OS displays an error message	
Primary Actors	User	
Trigger	User touched the app's icon	
Description	**Step**	**Action**
	0	The app is launched
	1	The initial screen is displayed
	2	The app shows the number of events on a selected date.
Extensions	**Step**	**Branching Action**
	1	*The app doesn't find any previous entries so the calendar's cells are not marked.*

Use Case 1	**Enter Data**	
Goal In Context	User will create a new calendar event	
Preconditions	User has successfully launched the application.	
Success End Condition	User has successfully created the new calendar event	
Failed End Condition	User failed to create a new calendar event	
Primary Actors	User	
Trigger	User touched a preferable date from the screen or the plus button at the bottom to add the new event	
Description	**Step**	**Action**
	0	The application launches the phone's calendar application
	1	User fill in the fields
	2	The app stores the entry to memory
	3	User returns to the initial screen
Use Case 2	**Start Calculation**	
Goal In Context	Find the optimal route	

Preconditions	User has entered all the required fields for the calculation to start	
Success End Condition	A map will be displayed with the optimal route	
Failed End Condition	Nothing will be displayed	
Primary Actors	Application	
Trigger	After the user has entered the required fields then the app automatically activates the navigation function	
Description	**Step**	**Action**
	1	The application acquires current location via Google servers
	2	The locations are sent to the server and for each one it finds the optimal route
	3	From the routes that the server sends back to the phone it chooses the fastest one.
	3.1	The application after it has collected all the data it shows the map with the optimal route.
Extensions	**Step**	**Branching Action**
	1	*Check scheduled times* After the app has received the data for the optimal route it compares the time given from the user with the estimated route duration and checks if he can stick to the schedule.
	1.1	*The user cannot make it in time so a pop up message appears*

		The user is informed that he has to leave in the next 15 minutes if he wants to make it on time.
	1.2	The user can make it in time
		We move on to the Map
	2	*On the Map the app checks if Gps is active*
	2.1	*GPS is active*
		The app shows the Map without the user knowing anything about this background check.
	2.2	*Gps is not active*
		A message pops up which asks the user to activate the Gps
	2.2.1	*User accepts*
		The app redirects him to the phone settings to activate his Gps and from there he can return
	2.2.2	*User declines*
		The app shows the Map and the navigation can begin

Use Case 2	**Following the current route**	
Goal In Context	Follow the route until the end of it	
Preconditions	User has entered all the required data, internet connection available,(Gps for better precision), location acquired and optimal route calculated.	
Success End Condition	The user will complete the route and the final message will appear	
Failed End Condition	User will never complete the route, no final message or maybe crash of the app	
Primary Actors	Application, User	
Trigger	After the map is displayed the user starts moving and his move is displayed on the map	
Description	**Step**	**Action**
	1	The app via Google's server tracks the position of the user while he is moving
	2	The application marks the areas visited once the user arrives at them
	3	The navigation ends when all places are marked as visited
Extensions	**Step**	**Branching Action**
	1	*The app checks the time of the events and compares it with current time and the remaining time for the next task*
	1.1	*User is running late* Pop up message appear notifying the user he is running late and disappears after a few seconds.
	1.2	*User is within time limits*

		User continues his route
	2	*App checks if the user is in a marked area*
	2.1	*User is not in a marked area*
		User continues his route
	2.2	*User is in a marked area*
		Alert the user about it with a pop up message
	2.2.1	*Application checks if there are any messages for this area*
	2.2.1.1	*There are messages for this area*
		A message pops up for this area for a short period of time and then the app returns to the map screen with the marked area. Eg Remember to buy staff from the supermarket
	2.2.1.2	There are no messages for this area
		The app just shows the map with the marked area
	3	*The user decided to change his mind and alter his route*
		The user presses the button alter in the menu
	3.1	A message pops up asking the user if he wants to choose one of the existing locations
	3.1.1	*User wants to go to one of the existing locations*
		A list of the existing locations is displayed and user chooses one of them and we recalculate the whole algorithm
	4	*The application checks how many locations are visited*

	4.1	*The number of visited locations equals the number of total locations so the user has completed his route and the application alerts the user with a message and exits the*
	4.2	*Still the number of locations visited is smaller than the total locations so the user continues his route, everything stays as it is, continue to check in background without disturbing the user.*

7.3 Behavioral Requirements

OS	Android
VERSION	2.2, 2.3, (compatibility with higher versions may be possible but not tested)
TESTING DEVICE	Huawei Ideos X5
DEVICE'S OS	Android 2.3
APP'S SIZE	108KB
RAM REQUIREMENTS	<15MB
CACHE	400KB

Chapter 8: CONCLUSIONS

This work is an effort to take advantage of the modern capabilities that smartphones offer in order to make our life easier. Developing this app is a multi-level procedure, starting from internet research about similar work not necessarily on the field of app programming, programming in different languages (java, xml), interface designing, algorithmic approach in order to give the app some functionality and accessing to phone's hardware components. By combining all of the above we have ourselves a working application environment.

REFERENCES

[1] http://developer.android.com/guide/topics/manifest/uses-permission-element.html

[2] http://developer.android.com/guide/components/activities.html#StartingAnActivity

[3] http://developer.android.com/reference/android/widget/Adapter.htMl

[4] http://stackoverflow.com/questions/6501413/how-to-create-dynamically-numbered-pin-pointers-on-mapview

[5] http://www.youtube.com/watch?v=qHAnGcnvRBI

[6] http://w2davids.wordpress.com/android-simple-calendar/

[7] http://www.youtube.com/watch?v=XdduYAs7klY

[8] http://developer.android.com/training/basics/firstapp/building-ui.html

[9] http://code.google.com/p/android-calendar-view/source/browse/trunk/src/com/exina/android/#android%2Fcalendar

[10] http://developer.android.com/training/basics/firstapp/creating-project.html

[11] Bui, T. N. and Zrncic, C. M. 2006. An ant-based algorithm for finding degree-constrained minimum spanning tree. In GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation, pages 11–18, New York, NY, USA. ACM.

[12] Garey, Michael R.; Johnson, David S. (1979), Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman, ISBN 0-7167-1045-5. A2.1: ND1, p. 206.

[13] Fürer, Martin; Raghavachari, Balaji (1994), "Approximating the minimum-degree Steiner tree to within one of optimal", Journal of Algorithms 17 (3): 409–423, doi:10.1006/jagm.1994.1042

[14] http://developer.android.com/sdk/exploring.html

[15] http://androidforums.com/application-development/292512-get-time-between-two-geopoints.html

[16] http://developer.android.com/about/versions/android-2.3-highlights.html

[17] http://cs.stackexchange.com/questions/1749/dijsktras-algorithm-applied-to-travelling-salesman-problem

[18] http://stackoverflow.com/questions/4373074/how-to-launch-android-calendar-application-using-intent-froyo

[19] http://jimblackler.net/blog/?p=151&cpage=2#comments

[20] http://developer.android.com/guide/topics/ui/declaring-layout.html

[21] Gutin, G.; Yeo, A.; Zverovich, A. (2002), "Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP", Discrete Applied Mathematics 117 (1–3): 81–86, http://www.sciencedirect.com/science/article/pii/S0166218X01001950

[22] http://stackoverflow.com/questions/5368769/open-and-display-calendar-event-in-android?rq=1

[23] http://developer.android.com/guide/topics/security/permissions.html

[24] http://blog.synyx.de/2010/06/routing-driving-directions-on-android-part-1-get-the-route/

[25] http://blog.synyx.de/2010/06/routing-driving-directions-on-android-–-part-2-draw-the-route/

[26] http://developer.android.com/training/basics/supporting-devices/platforms.html

[27] https://developers.google.com/maps/documentation/distancematrix/#JSON

[28] http://www.openhandsetalliance.com/android_overview.html

[29] http://officialandroid.blogspot.gr/2012/09/google-play-hits-25-billion-downloads.html

[30] http://developer.android.com/about/dashboards/index.html

[32] http://developer.android.com/sdk/installing/adding-packages.html

[33] http://developer.android.com/sdk/installing/installing-adt.html

[34] http://developer.android.com/training/basics/firstapp/creating-project.html

[35] http://developer.android.com/training/basics/firstapp/running-app.html

[36] http://developer.android.com/training/basics/firstapp/building-ui.html

[37] http://developer.android.com/training/basics/supporting-devices/screens.html

[38] http://stackoverflow.com/questions/11323500/google-maps-api-version-difference/11357351#11357351