DESIGN AND IMPLEMENTATION OF MOBILE PHONE APPLICATION (CAMPUS GUIDE)

BY

AYANWALE ABDULLATEEF .A.

2004/18795EE

ELECTRICAL AND COMPUTER ENGINEERING DEPARTMENT

FEDERAL UNIVERSITY OF TECHNOLOGY

MINNA NIGER STATE

DECEMBER, 2009

DESIGN AND IMPLEMENTATION OF MOBILE PHONE APPLICATION (CAMPUS GUIDE)

BY

AYANWALE ABDULLATEEF .A.

2004/18795EE

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIRMENT FOR THE AWARD OF BACHELOR OF TECHNOLOGY (B.ENG) DEGREE IN THE DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING.

SCHOOL OF ENGINEERING AND ENGINEERING TECHNOLOGY (SEET)

FEDERAL UNIVERSITY OF TECHNOLOGY, MINNA.

DECEMBER, 2009.

DEDICATION

This work is dedicated to Allah, who has been the source of my inspiration, the source of all knowledge and understanding in this world and the hereafter. Also to my father Ayanwale Imuran and my loving mother whose moral and financial contribution cannot be overestimated. May almighty Allah reward them both with Jannatul Firdausi ameen.

DECLARATION

I, Ayanwale Abdullateef Adekunle, declare that this work was done by me and has never been presented elsewhere for the award of a degree. I also hereby relinquish the copy right to the Federal University of Technology, Minna.

Ayanwale Abdullateef .A.
(Student)

15/12/2009
(Signature and Date)

Engr. (Mrs.) C.O. Alenoghena
(Supervisor)

18/12/09
(Signature and Date)

for

Engr. Dr. Y.A Adediran
(Head of Department)

(May 6, 2010)
(Signature and Date)

External Examiner

3/03/10
(Signature and Date)

ACKNOWLEDGEMENT

My utmost gratitude goes to Allah, my provider and sustainer. He gives me life and makes everything possible for me. To my mother Mallama Kudirat Ayanwale for the greatest support, my sincere gratitude also goes to my siblings Kehinde, Taiye, Olamide, and Tope for their general assistance in all my life endeavors.

I am much indebted to my supervisor; Engr. (Mrs.) C.O. Alenoghena for her support in this work. She makes me work hard, for her constructive criticism, technical assistance and guidance in the actualization of this project. My profound gratitude goes to all my lecturers for the knowledge that materialized into this project.

I will not forget the tremendous help from my colleagues in school such as Ibrahim Wasiu, Adam Salihu, Ayuba Emmanuel, Marwan Abubarar, Bima Muhammed and others too numerous to mention. God bless you.

ABSTRACT

The mobile phone application (GIDA GKANU CAMPUS-GUIDE) explore J2ME, the Java platform for small devices, a broad field that covers pretty much everything smaller than a breadbox. Object orientated language was used in developing the application. The project is aimed at making navigation and movement easy for everybody on the campus, creating source of revenue for the department as well as to demonstrate how software application can be embedded in mobile phone. It is compatible with most application phone, capable of being visitor guide by showing the major landmarks of the campus. Consultation with the planning unit of the school to get comprehensive landmarks map after which the map was studied for proper analysis before the real programming was carried out. The performance of the design and implementation of this project met the design specifications. It accomplished the aims and objectives of the project, which is to implement a Campus Guide on mobile phone.

Table of Contents

CHAPTER ONE

1.1 INTRODUCTION

Software engineering is the design and implementation of software programs on computer systems and other compatible devices. Software as the name connotes, is a non physical component of a programmed system which orders and directs the execution of tasks based on the system design. It is in fact a set of instruction systematically compiled to execute specific tasks or carryout certain activities. Software programs could be written as code and executed as stand-alone programs on a computer system or can be interfaced with other compatible electronic devices such as microcontrollers, microprocessors, other programmable electronic chips or mobile phone. [1]

J2ME is the Java platform for small devices. a broad field that covers pretty much everything smaller than a breadbox. Because J2ME spans such a diverse selection of hardware, it is divided into configurations, profiles, and optional APIs. A configuration specifies a subset of J2SE functionality and the behavior of the JVM, while profiles are generally more specific to a family of devices with similar characteristics. Optional APIs offer added functionality in a flexible package. The Mobile Information Device Profile, which is the focus of this project, includes APIs for devices like mobile phones and two-way pagers. [2]

The **"Campus Guide"** is a Mobile Phone Application Software designed to be compatible with any application mobile phone and even Black-Berry phone. The software is design for Federal University of Technology, Minna to make available the landmarks of the main campus for easy navigation of visitors.

1.2 AIMS AND OBJECTIVES

- To easy the navigation of everybody on the campus
- To serve as source of revenue for the department.
- To demonstrate how software application can be embedded in mobile phone.

1.3 METHODOLOGY

In the quest of designing and developing this mobile phone application, the following steps were taken:

- Consultation of the planning unit of the university to get the map of the campus landmark.
- A careful study and analysis of the map was under taken so as to have a clear understanding of the campus terrain.
- Decision on which programming language and other programming tools that will be best suitable for the development.
- Designing a program flow chart (otherwise known as **activity diagram**) to serve as a guide on writing the program code.
- Engage in writing the program code to execute the different stages and tasks of the software.
- Debugging and testing (against the specification) of the software

1.4 SCOPE AND LIMITATION OF PROJECT

The mobile phone application will be capable of the following:

- It will be compatible with any application mobile phone such as Nokia, Samsung, Motorola, Sony Erikson and even Black berry.
- The application will be capable of showing the landmark of Gida Gkano Campus.
- Capable of being a visitor guide on the campus.
- It allows the pre-setting of desired destination by user.
- It prompts the user upon reaching the set destination.

"The Campus Guide will alert you whenever you get close to landmark of selected category."

The mobile phone application is limited to the following:

- It cannot support phone without flash memory e.g. the so called ***Chinese*** phone.
- It is limited to cover only the important landmarks of Gida Gwankanu campus.

1.5 PROJECT OUTLINE

Chapter one gives a general introduction of what the project is all about, its aims and objectives, methodology and scope of the project.

The previous work done with respect to this project is discussed as the literature review of chapter two.

Chapter three covers the various steps taken in the design and implementation of the project wok and calculations involve.

The tests carried out, result obtained and discussion of such results come under chapter four.

The conclusion and recommendation constitutes chapter five.

CHAPTER TWO

LITERATURE REVIEW

2.1 HISTORICAL BACKGROUND

Navigation is the process of reading, and controlling the movement of a craft, vehicle or human being from one place to another. It is also the term of art used for the specialized knowledge used by navigators to perform navigation tasks. The word *navigate* is derived from the Latin "navigare", meaning "to sail". All navigational techniques involve locating the navigator's position compared to known locations or patterns.

Latitude

The latitude of a place on the earth's surface is the angular distance north or south of the [equator]. Latitude is usually expressed in degrees (marked with °) ranging from 0° at the Equator to 90° at the North and South poles. The latitude of the North Pole is 90° N, and the latitude of the South Pole is 90° S. Historically, mariners calculated latitude in the Northern Hemisphere by sighting the North Star Polaris with a sextant and sight reduction tables to take out error for height of eye and atmospheric refraction. Generally, the height of Polaris in degrees of arc above the horizon is the latitude of the observer.

Longitude

Similar to latitude, the longitude of a place on the earth's surface is the angular distance east or west of the prime meridian or Greenwich meridian. Longitude is usually expressed in degrees (marked with °) ranging from 0° at the Greenwich meridian to 180° east and west. Sydney, Australia, for example, has a longitude of about 151° east. New York City has a longitude of about 74° west. For most of history, mariners struggled to determine precise longitude. The problem was solved with the invention of the marine chronometer. Longitude can be calculated if the precise time of a sighting is known.

2.1.1 NAVIGATION INSTRUMENTS

There are several categories of navigation instruments, among which are:

1. **Charts and drafting instruments**

- Charts are maps of the areas to be navigated with details specific to the marine environment.
- **Computing aids**: used in the necessary mathematical calculations. Today electronic computers or calculators are used. Other traditional aids used included tables (trigonometric, logarithms, etc) and slide rules.
- **Dividers** used for measuring lengths of lines and approximate lengths of non-linear paths on a chart.

- **Nautical almanac** used to determine the position in the sky of a celestial body after a sight has been taken.
- **Parallel rules** used for transferring a line to a parallel position. Also used to compare the orientation of a line to a magnetic or geographic orientation on a compass rose.
- **Pencil**, **straight edge** and other conventional drawing tools.

2. **Direct measuring**

- **Chip log and sand glass** serve to measure the ship's speed through the water
- **Sounding line** used to measure the depth of the water and to pick up samples from the bottom

3. **Position finding instruments**

These instruments are used primarily to measure the elevation or altitude of a celestial object:

- **Back staff**, the best known of which is the Davis' quadrant. It could measure the altitude of the sun without having the navigator directly observe the sun.
- **Cross staff**, an older instrument long out of use.
- **Kamal** Very simple instrument used primarily by Arabian navigators. It consists of a small board with a knotted piece of twine through the center. The observer holds one of the knots in his mouth and extends the board away so that the edges make a constant angle with his eyes.

- **Mariner's astrolabe** Derived from the astrolabe, it was developed in late 15th century and found use in the 16th to 17th centuries. It was replaced by the back staff and later by the octant and sextant.
- **Quadrant** A very simple instrument which used a plumb bob.

These instruments are also used to measure the angular distance between objects:

- **Octant**, invented in 1731. The first widely-accepted instrument that could measure an angle without being strongly affected by movement.
- **Sextant**, derived from the octant in 1757, eventually made all previous instruments used for the same purpose obsolete.

4. **Bearing instruments**

- **Pelorus** used to determine bearings relative to the ship's heading of landmarks, other ships, etc.

5. Compasses

- **Bearing compass** used to determine magnetic bearings of landmarks, other ships or celestial bodies.
- **Magnetic compass** used to determine the magnetic heading of the ship

6. **Timekeeping**

- **Marine chronometer** used to determine time at the prime meridian with great precision which is necessary when reducing sights in celestial navigation

- **Nocturnal** used to determine apparent local time by viewing the Polaris and its surrounding stars.
- **Ring dial** or **astronomical ring** used to measure the height of a celestial body above the horizon. It could be used to find the altitude of the sun or determine local time. It let sunlight shine through a small orifice on the rim of the instrument. The points of light striking the far side of the instrument gave the altitude or tell time.

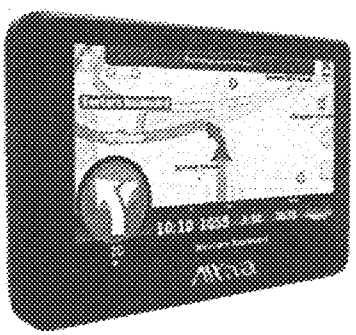
All those mentioned were the traditional instruments used until well into the second half of the 20th century. After WWII electronic aids to navigation developed very rapidly and, to a great extent, replaced more traditional tools. Electronic speed and depth finders have totally replaced their older counterparts. Radar has become widespread even in small boats. Some Electronic aids to navigation like Loran have already become obsolete themselves and have been replaced by GPS. [3]

2.2 PREVIOUS WORKS

Over the years, much innovative work has been carried out in making the navigation of human being across the globe easier as demonstrated by people embarking on various research work, among which are:

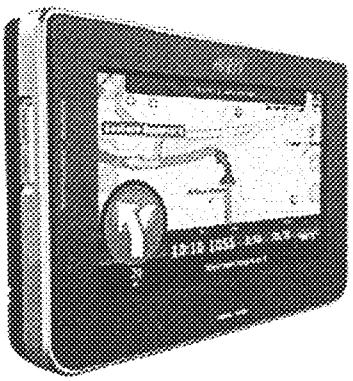
Research on navigation of autonomous mobile robot with goal direction detection sensor (Electro Mechanic Technology Advancing Foundation (EMTAF); sponsor)

Self position estimation sensor system (goal direction detection system) was developed to apply to autonomous mobile robot. The system plays an important role when the robot moves within indoor workshop. The system consists of four units of infrared ray photo-detectors, DC motor, rotation angle potentiometer and infrared light emission sources (landmark), and is controlled in real time. As a result of test, it was found that though the robot accurately moves within 5 m of distance from landmark, it becomes inaccuracy within 8 m of the distance, and hence guide system with multiple landmarks is required. YAMAMOTO MOTOJI (Kyushu Univ., Graduate School of Engineering, JPN) is the researcher of this work.



Altina G8010 with NDrive

Latest 4.3 inch wide screen GPS technology, running NDrive navigation.

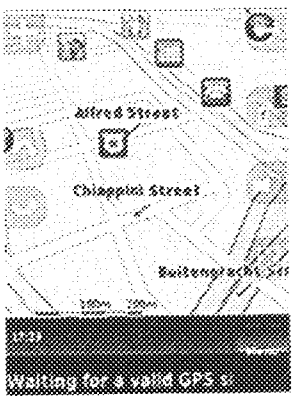


AIRIS T940 with NDrive

4.3inch wide screen GPS with Bluetooth hands-free mobile phone operation and NDrive navigation.

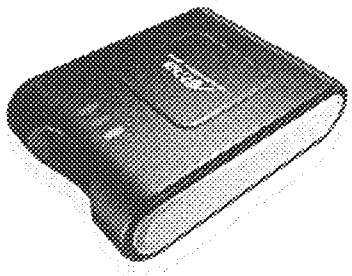
GPS TRACK LOGGER

Low cost, match box sized GPS logger, displays where a vehicle has driven & stopped on Google Earth. Used in aviation for flight tracking.



NDrive Navigation Software

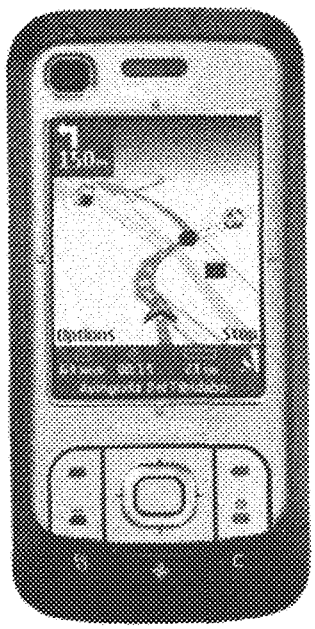
The most advanced, fastest and smoothest navigation software, on SD card for pocket PCs and phones using Windows Mobile 5 or 6.



Altina Bluetooth GPS, GBT-709

Match box sized GPS receiver for use with Bluetooth enabled devices.

The Nokia 6110 navigator manufactured by the mobile phone producer giant was also produced to ai gation. It uses the GPS technology as its operational means of locating landmarks. It is capable of showing th marks of major cities in the world, the various cities longitude and latitude are already configured with the phone.



CHAPTER THREE

DESIGN AND IMPLEMENTATION

3.1 SPECIFICATION & SOFTWARE OVERVIEW

It is often difficult (or even impossible) to obtain a set of unambiguous, fixed requirement for a software system although there are a variety of reasons for this, I assume a sufficient familiarity with the operation of mobile phone to understand the following description:

CONNECTED, LIMITED DEVICE CONFIGURATION

CLDC is the configuration that is implemented in this project, because it encompasses mobile phones, pagers, PDAs, and other devices of similar size. The name CLDC appropriately describes these devices, having limited display, limited memory, limited CPU power, limited display size, limited input, limited battery life, and limited network connection.

The CLDC is designed for devices with 160KB to 512KB of total memory, including a minimum of 160KB of ROM and 32KB of RAM available for the Java platform. The "Connected" simply refers to a network connection that tends to be intermittent and probably not very fast. (Most mobile telephones, for example, typically achieve data rates of 9.6Kbps.) These connections also tend to be costly, typically billed by the data packets exchanged. Between the high cost and intermittent slow network connection, this applications design in the CLDC space is intended to be very sparing with the use of the network connection.

The reference implementation of the CLDC is based around a small JVM called the KVM. Its name comes from the fact that it is a JVM whose size is measured in kilobytes rather than megabytes. While the CLDC is a specifications document, the KVM refers to a specific piece of software.

1. Because of its small size, the KVM can't do everything a JVM does in the J2SE world.
2. A mobile phone application, compactable with any application phone such as Nokia, Samsung, Motorola, Sony Erikson and Black Berry is the task at hand.

3.2 THE APPLICATION DESIGN

In the process of developing this software, a top-down process was employed, in which the sequence is a progression from the general to the specific. The process is categorized into a number of identifiable segments as follows;

1. problem Analysis
2. Software specification
3. Development of algorithm
4. Program definition
5. Building the application (Coding)
6. Testing (against the specification)
7. Debugging
8. Implementation
9. Evaluation

10. Ongoing maintenance

The above list constitutes an outline that constitutes phases in the software development cycle of this application development. [5]

The application development and design involves the development of an algorithm which addresses conceptualizing the means of solving the problem at hand. The algorithm development is achieved by means of a flowchart, which involves breaking down the problem into a number of smaller steps or processes.

3.2.1 Program definition (using unified modeling language, uml)

The object-oriented system used comprises a number of software objects that interact to achieve the system objective. The software objects in this design mimic the real-world objects of the application domain. The real-world objects may have a physical presence or may represent some well-understood conceptual entity in the application. In the campus guide application there is software objects that represent landmarks. Equally, there are some software objects representing various directions of landmarks even though they have no physical existence.

The Objects are characterized by having both *state* and *behaviour*. The state of an object is the information an object has about itself. For example, a landmarks object has a name, location, shape, size and a colour. Equally, a direction of landmarks object has the name of the direction, its distance, longitude and latitude. The behaviour of the objects describes the actions the objects are prepared to engage in. The behaviour of the object is described by the set of *operations* it is prepared to perform. [6]

One object interacts with another by asking it to perform one of its advertised operations. This interaction is achieved by one object sending a *message* to another. The first object is known as the *sender object* and the second object is known as the receiver or *recipient object*. The only messages the objects can receive in this application belong to the set of operations they can accept. In the UML, objects and message passing are captured by a *sequence diagram* which described below.

1. **Use-case diagram**

   The *use-case* shows a typical interaction between a user and the system under development.

   This is used to capture some functionality to be provided by the software system. For example, in a banking application a use-case may document that a requirement of the system is to support transactions on bank accounts, e.g. make a deposit. Similarly, in this application setting the category of desired destination is presented as a use case.

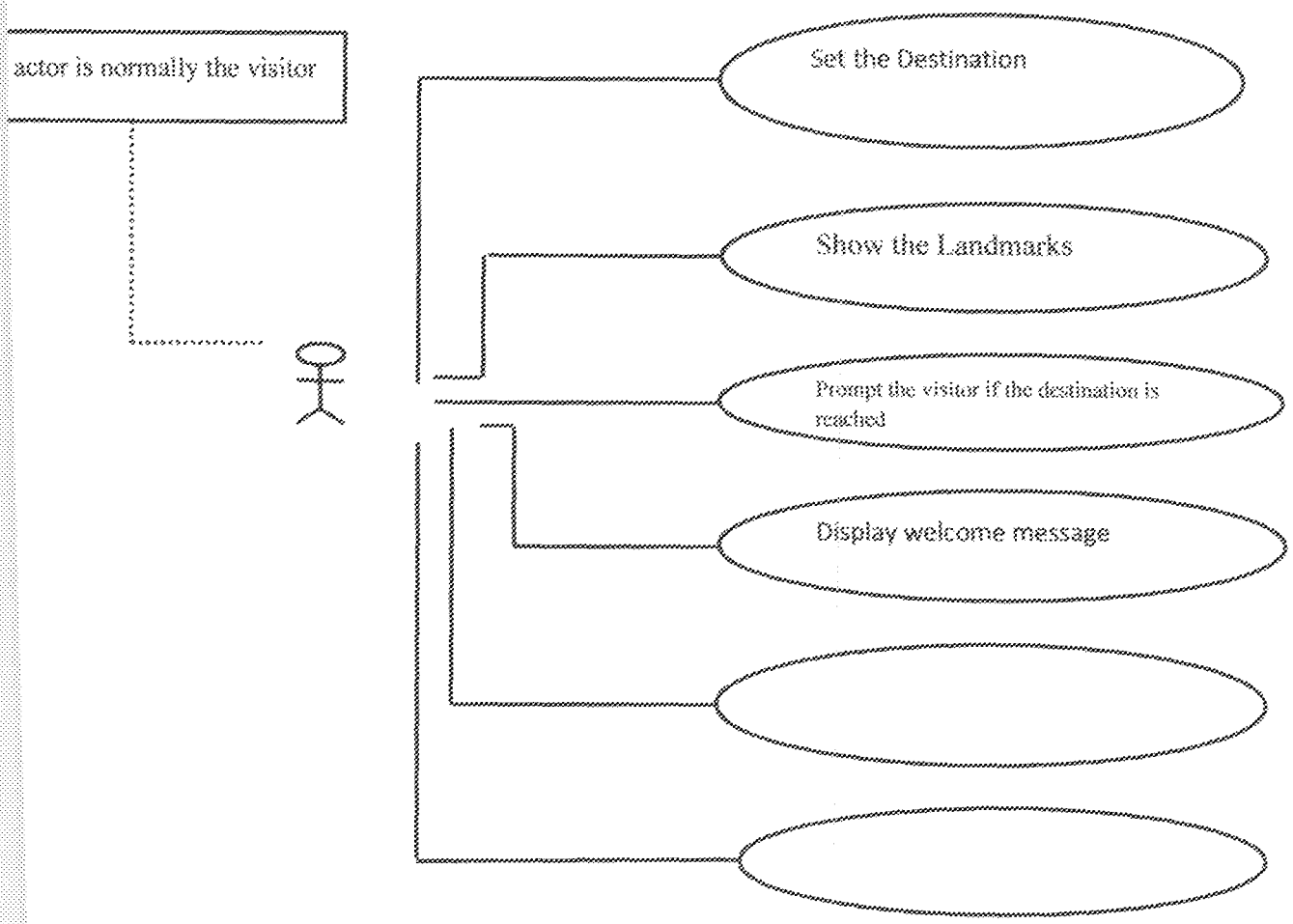


Fig. 3.1 The use-case Diagram

2. **Interaction diagram**

The aim of the interaction diagram is to describe how groups of objects collaborate to achieve some behavior. I employ an interaction diagram to capture the behavior for a single use-case. Both sequence and collaboration diagrams present a number of objects and the messages that are passed between them.

3. **Sequence diagram**

In the sequence diagram the objects are presented as rectangular figures. In figure 2.5(c), for example, we show two object instances: a Landmarks object and Direction object. The object instances are decorated with a label of the form lm: Landmarks, where lm is the object's identifier and Landmarks is the class (or type) of the object.

4. **Collaboration diagram**

Collaboration diagrams use the same mix of actors and instance symbols as found in sequence diagrams. A *link* between two object instances is shown by a connecting line.

The link is decorated with numbered messages sent between the instances. The relationship shows whether the instances are associates or are aggregates. Like the sequence diagrams I also annotate messages in a collaboration diagram with conditions and iterations.

5. **Activity diagram**

The activity diagram is used to model various dynamic aspects of the system. At its simplest, the activity diagram presents the flow of control between activities.

The activity diagram focuses on the separate activities within the system, whereas interaction diagrams are concerned with the execution flow through the network of objects. The activity diagrams were used to document use-cases as well as the behavior of methods.

The essence of the activity diagram is an *action* that represents some executable computation.

They are called *action states* and are represented by soft boxes populated with text describing the action performed. Upon completion of one action, control automatically proceeds to the next along a directed transition.

6. **Class diagram**

The *class diagram* is the principal diagram that is constructed in this object-oriented design. Its importance lies in the fact that its content delivers the primary elements in this program code, namely the Java classes. The class diagram describes the types of the objects in the system and the relationships that exist between them. The class diagram is an abstraction for all the possible object diagrams that were constructed. Class diagrams and collaboration/object were made to be consistent. since a configuration of objects can exist, then the class diagram needs to capture this information. The class diagram is used to shows that two object types are unrelated by the absence of link between corresponding instances in the collaboration or object

diagram. It also means that a message cannot be shown from one object to another in the sequence diagram.

Each class is also documented with its set of attributes and operations. The attributes represent the set of values each instance maintains as the object's state. The set of operations is the messages an object of the class may receive. Some of these operations can be identified from the various interaction diagrams that were constructed.

The software development "Campus Guide" follow the design sequence as represented below.

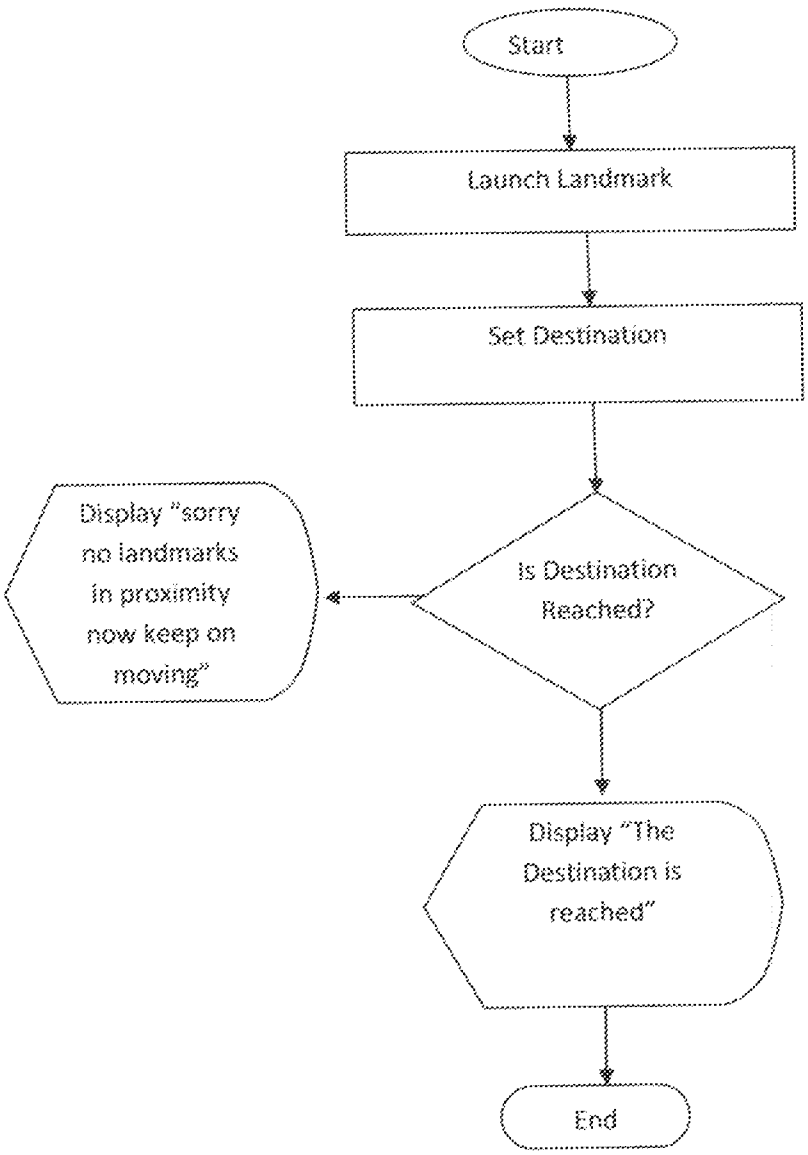


Fig. 3.2 Flow Chart of the Application

3.2.2 Building the application

The MIDP application which is piquantly called MIDlets.

The actual development process, however, is a little more complicated for MIDlets than it is for J2SE applications. Beyond a basic compile-and-run cycle, MIDlets require some additional tweaking and packaging. The complete build cycle looks like this: Edit Source Code ➢ Compile ➢ Preverify➢ Package ➢ Test or Deploy. [7]

Creating Source Code

Writing Java source code is the same as it always was: using favorite text editor to create a source file with a .java extension.

```
package examples.cityguide;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.util.Enumeration;

import java.util.Hashtable;

import java.util.NoSuchElementException;

import java.util.Vector;

import javax.microedition.lcdui.*;
```

```
import javax.microedition.lcdui.List;
import javax.microedition.location.AddressInfo;
import javax.microedition.location.Coordinates;
import javax.microedition.location.Criteria;
import javax.microedition.location.Landmark;
import javax.microedition.location.LandmarkStore;
import javax.microedition.location.LocationException;
import javax.microedition.location.LocationProvider;
import javax.microedition.location.QualifiedCoordinates;
import javax.microedition.midlet.*;

/**
 * Main CampusGuide MIDlet
 */
public class CampusGuideMIDlet extends MIDlet implements CommandListener {
    /** landmark store name to store points of interest */
    private static final String LANDMARKSTORE_NAME = "waypoints";
```

```
/** welcome text for welcome screen */
  private static final String WELCOME_TEXT =
    "Welcome to The Campus Guide. First JSR179 enabled campus guide." + "
Select your favorite landmark categories and never miss interesting place while
walking through the city." +  " Don't forget to run the citywalk.xml script\n\n" +
"Enjoy the show!\n\n";
/** explanation label for settings screen */
  private static final String SETTINGS_TEXT =
    "The Campus Guide will alert you whenever you get close to landmark of
selected category."
/** choice group label for categories */
  private static final String CHOICEGRP_TEXT = "Watch for categories:";
/** no landmark in proximity */
  private static final String NOLANDMARK_TEXT =
    "Sorry no landmark in proximity now.\nKeep on walking.";
/** map corners coordinates */
  private static final Coordinates
MAP_TOP_LEFT_COORDINATES =
```

```
new Coordinates(14.387594174164514, 50.1049422484619, 310);

private static final Coordinates

MAP_BOTTOM_RIGHT_COORDINATES =

    new Coordinates(14.404239767000292, 50.09531507039965, 310);


  /** initial visitor coordinates */

  private static final Coordinates MAP_VISITOR_COORDINATES =

    new Coordinates(14.389796708964603, 50.09985002736201, 310);

  private static final String[] PRELOAD_IMAGES =

    {

      "map", "visitoron", "visitoroff", "anim1", "anim2", "anim3", "anim4",
"anim5", "anim6",

      "anim7", "anim8", "logo"

    };

  private static CityGuideMIDlet instance;

 /** error screen uses last error occurred */

  private String lastError = "";
```

```
/** list of selected categories */
  private Vector selectedCategories;

  /** landmark store containing points of interest */
  private LandmarkStore landmarkStore = null;
/** location provider */
  private LocationProvider locationProvider = null;
/** distance when proximity event is sent */
  private float proximityRadius = 50.0f;
/** image cache */
  private ImageManager imageManager = null;
/** landmarks successfully loaded */
  private boolean landmarksLoaded = false;
/** main MIDlet display */
  private Display display;
/** map canvas */
  private MapCanvas mapCanvas;
```

```
/** city map */
  private CityMap cityMap;
 /** Screens and commands */
  private Command WelcomeScreen_nextCommand;
  private Command ProgressScreen_nextCommand;
  private Command showErrorCommand;
  private Command SettingsScreen_backCommand;
  private Command DetailsScreen_backCommand;
  private Command exitCommand;
  private Command closeCommand;
  private final Command detailsCommand;
  private final Command settingsCommand;
  private Gauge progressGauge;
  private Form progressScreen;
  private Form welcomeScreen;
  private Form settingsScreen;
  private Form detailsScreen;
```

```
private Alert errorAlert;


/**
 * Main Campus Guide MIDlet
 */
public CampusGuideMIDlet() {
    exitCommand = new Command("Exit", Command.EXIT, 1);
    closeCommand = new Command("Close", Command.SCREEN, 1);
    showErrorCommand = new Command("Error", Command.SCREEN, 1);
    detailsCommand = new Command("Detail", Command.SCREEN, 1);
    settingsCommand = new Command("Settings", Command.SCREEN, 1);
    display = Display.getDisplay(this);
    imageManager = ImageManager.getInstance();
    imageManager.getImage("logo");
    createLocationProvider();


    if (locationProvider == null) {
```

```
            System.out.println("Cannot run without location provider!");
            destroyApp(false);
            notifyDestroyed();
        }
instance = this;
    }
    public static CityGuideMIDlet getInstance() {
        if (instance == null) {
            instance = new CityGuideMIDlet();
        }
        return instance;
    }
    public void startApp() {
        if (display.getCurrent() == null) {
            // startApp called for the first time
            showWelcomeScreen();
        } else {
```

```
        if (cityMap != null) {
            cityMap.enable();
        }
    }
}
```

Compiling the MIDlet

Writing MIDlets is an example of cross-compiling, where you compile code on one platform and run it on another. In this case, MIDlet was compiled using J2SE on desktop computer. The MIDlet itself will run on a mobile phone, pager, or other mobile information device that supports MIDP.

The J2ME Wireless Toolkit takes care of the details as long as the source code is in the right directory.

1. Start the toolkit, called KToolbar.

2. Choose New Project from the toolbar to create a new project.

3. When the J2ME Wireless Toolkit asks you for the name of the project and the MIDlet class name, use "Jargoneer" for both.

4. Click the Create Project button, and then the OK button to dismiss the project settings window.

Preverifying Class Files

Now comes an entirely new step in building your program, *preverifying*. Because the memory on small devices is so scarce, MIDP (actually, CLDC) specifies that bytecode verification be split into two pieces. Somewhere off the device, a preverify step is performed. The device itself is only required to do a lightweight second verification step before loading classes.

If you are using the J2ME Wireless Toolkit, you don't have to worry about preverifying class files, and you may not even notice that it's happening when you click the Build button. If you'd like to understand more about preverifying, read the rest of this section. Otherwise, you can just skip ahead.

As you may recall, bytecode verification is one of the foundation stones of Java's runtime security model. Before a classloader dynamically loads a class, the bytecode verifier checks the class file to make sure it behaves well and won't do nasty things to the JVM. Unfortunately, the code that implements the bytecode verifier is bulky, too large to fit on a small device like a mobile phone. The CLDC dictates a two-step bytecode verification:

1. Off the device, class files are preverified. Certain checks are performed, and the class file is massaged into a format that the lightweight second-step verifier can easily handle. This format is really just a regular class file, with some additional data attached by the preverifier.

2. On the device, the second step of verification is performed as classes are loaded. If a class file has not been preverified, it is rejected.

CHAPTER FOUR

TEST, RESULT AND DISCUSSION

4.1 TESTING

RUNNING JMunit TESTS

A test was generated by selecting a class and packages node in the project window and choosing from the Tool > JUnit menu.

When the first test is generated, a TestSuite MIDlet from the framework JAR gets added to the list of MIDlet was shown in the emulator window's Run list when the project was running.

All test classes were added to the property in the JAD manifest called JMunitTestClasses. When the first test was generated for the project, the JMunit framework JAR gets added to the project libraries for all project configurations. The framework JAR builds a list of tests to run from the JNunitTestClasses property. The empty tests JMunit created were modified to include assertions of expected results to the pass/fail results relevance.

4.2 RESULT

When the software was installed on Nokia application supporting phone with GPRS (internet facilities), the landmark appeared on the phone showing different locations.

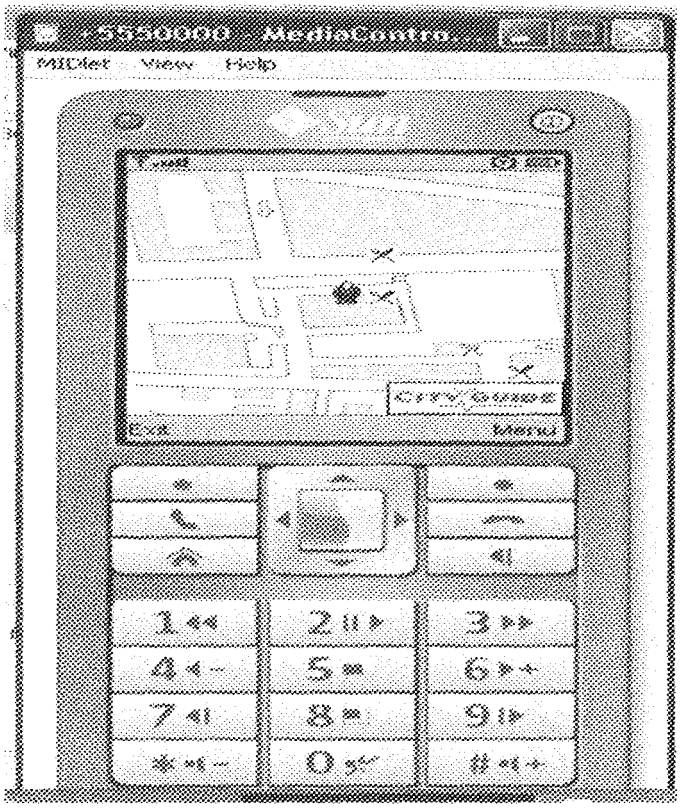


Fig. 4.1 Phone Showing the Operation of the Application.

4.3 DISCUSSION OF RESULT

When the application was launched and a destination was set the phone was displaying "keep on moving, the preset destination is not yet reached" whenever detailed is confirmed. As soon as the destination was reached it prompted that the destination is reached.

CHAPTER FIVE

5.1 CONCLUSION

The performance of the design and implementation of this project met the design specifications. It accomplished the aims and objectives of the project, which is to implement a Campus Guide on mobile phone.

5.2 RECOMMENDATION

I recommend that this application software should be made available on futminna web site for commercialization. Its availability on internet will ease accessibility and consequently increase its marketability and revenue generation.

REFERENCES

[1] Ivande Ephraim Terkuma, "Design & Implementation of Web Based intelligent Result Multicasting Software", Final Year Project, Department of Electrical & Computer Engineering, FUT, Minna, 2008.

[2] Sing Li and Jonathan Knudsen, Beginning J2ME, Third Edition , Novice to Professional, United States of America, ISBN (pbk): 1-59059-479-7.

[3] http://www.wikipedia.com/navigation-instrument, Retrieved November 28, 2009.

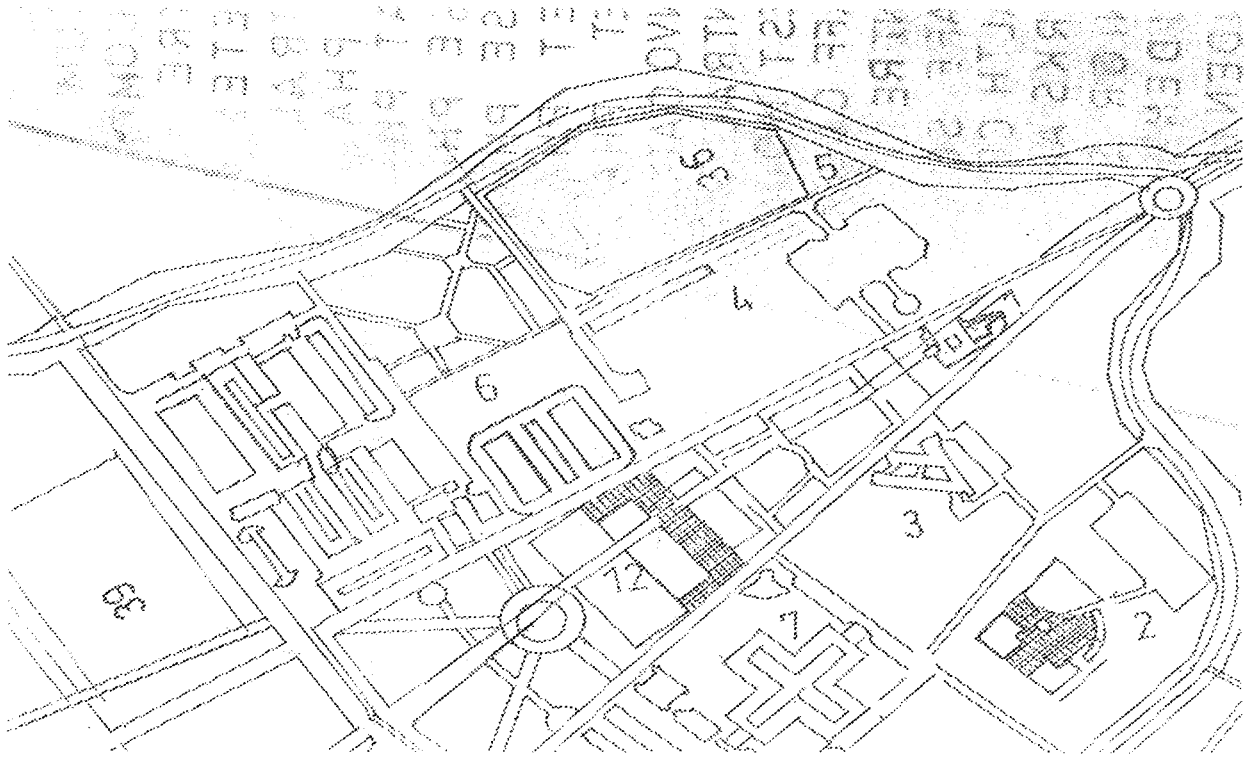
[4] http://www.citynavigation.com Retrieved November 28, 2009.

[5] Appears in Roger S. Pressman, *Software Engineering (A practitioner's approach)*, 5th edition, 2000, Mc Graw-Hill Education, ISBN 978-0071181822

[7] K. Barclay and J. Savage, Object-Oriented Design with UML and Java, Elsevier Butterworth-Heinemann, Linacre House, Jordan Hill, Oxford OX2 8DP.

APPENDICES

PROJECT FILE DETAILS		
S. No	**File Name**	**Remarks**
1	CampusGuideMIDiet.java	This is the class that holds the main method.
2	CampusMap.java	This is the class responsible for the calculation of longitude and latitude of various landmarks.
3	ImageManager.java	It is the class responsible for display of the landmarks
4	MapCanvas.java	It contains the canvas of the map.
5	MapLandmark.java	It points to the set landmarks.
6	MapListener.java	This is the class responsible for the sensitivity of the application
7	Util.java	It takes care of the ordering of the images



LANDMARKS OF GIDA GKANU CAMPUS, FUTMINNA.

LEGEND	
LIBRARY	1
RESTRAURANT	2
CENTRAL ADMINISTRATION	3
MULTIFUNCTIONAL CENTRE	4
CAFETERIA & CENTRAL	5