Karoshi

Initial Design Report

MAP-MET [Map-Military Enhancing Technology]

Fatma Akıncı
İsmail Can
Coşkuner
İlker Argın
Meryem Sağcan

27.12.2010

1 Introduction

1.1 Problem Definition

Most of the mobile devices, today, have a static user interface which does not respond to any environmental change or user's motion. One may have difficulties in seeing the screen content and using the device when light conditions change or he/she moves. When soldiers are taken into consideration, they are often in motion and they occur to be in places such that the light quantity in the environment is low.

Current map viewer applications, which run on mobile devices, used in military have static buttons, menus and texts whose sizes do not change. The colors of the map shown on the screen do not change, neither. This kind of static user interface of current map applications creates a big problem for the soldiers.

The final product will be a mobile map application with dynamic user interface and therefore will be a solution to difficulty of use of current map applications used for military purposes.

1.2 Purpose

This Software Design Document (SDD) aims to provide a description of the software product in order to give the developers a guidance of the architecture of the software. The document details how the software requirements should be implemented in a way that the structure of the system explained satisfies the requirements mentioned in Software Requirements Specification. The components of the product and their properties will be clearly explained.

1.3 Scope

This document contains a complete description of the design of MAP_MET. All the components and their functions of the product are explained in the document. The intended audiences are code developers of the product. Hence, this report will serve as a guideline throughout the development of the Project.

1.4 Overview

The following chapters and their contents are

- Chapter 2 is System Overview that includes a general description of the overall system and its design. The benefits and the differences of the product from the similar products are explained in this chapter.

- Chapter 3 is Design Considerations mentions the issues related to design. The constraints that affect the design of the architecture of the software and the use of the final product. Any design goals and principles that form the software of the system is also clarified here.
- Chapter 4 is the Data Design that lists any data used for the system to properly work. The data stored, managed, or manipulated are listed with their descriptions, types and attributes.
- Chapter 2 is a Deployment Diagram that shows the physical nodes on which the system resides. This allows a clear explanation of where each design entity will reside. No design unit may straddle two nodes but must have components on each, which collaborate to accomplish the service.
- Chapter 5 is the System Architecture. This is the heart of the document. It specifies the design entities that collaborate to perform the functionality of the system. Each of these expresses the services that it provides to the rest of the system. To clearly explain the design a component diagram for the software is drawn and all the packages and their classes in them are shown in the package diagrams.
- Chapter 6 explains the content of the user interface. It shows the screen view of the application and clarifies how each screen object functions. The screenshots of the user interface are also included in this chapter.
- Chapter 7 lists the libraries and tools that will be used in development process.
- Chapter 8 includes Gantt chart illustrating the start and finish dates of the terminal elements and summary elements of the project.
- Chapter 9 concludes the report.

1.5 Definitions, Acronyms and Abbreviations

SDD: Software Design Description

IDR: Initial Design Report

DDR: Detailed Design Report

MAP_MET: Map Military Enhancing Technology

Java ME: Java Micro Edition

1.6 References

[1] http://opencv.willowgarage.com/wiki/

[2] http://www.java.com/tr/

[3] http://www.mathworks.com/

[4] http://www.eclipse.org/

[5] http://qt.nokia.com/products/

[6] Kozierok, C. (2005). *The TCP/IP guide : a comprehensive, illustrated Internet protocols reference.* San Francisco: No Starch Press.

2 System Overview

MAP_MET is a kind of military map viewer application with additional functionalities. MAP-MET is capable of detecting user's motion and environment's light condition. This provides the system with adaptable map visualization system and dynamic user interface. The goals of such a product will be:

- Increase readability when user in motion by hiding details and rarely used buttons, enlarging font sizes, zooming in map, and remarkable coloring of important text.
- Increase visibility in light condition change by adjusting brightness and contrast according to illumination, adjusting colors of map, application background, buttons and text regarding their visibility of human visual system.

Moreover, our application can work on mobile device with a single camera and CPU that has limited computation power. Similar application that use camera needs powerful CPU to achieve image processing operations. Since we will do these complex operations in the server machine, our application can be used in wide variety of mobile devices.

3 Design Considerations

3.1 Design Assumptions, Dependencies and Constraints

3.1.1 Time Constraints

This project is a senior student project, given by the department of Computer Engineering. So the schedule and timing is determined and strict. After this report there will be a certain deadline for a final decision report and a prototype must be accomplished in a month. The main implementation of the project will be done in second term. A task distribution and the needed time duration for these tasks are decided in this document. Gantt chart of the project is given in the Project Schedule section.

3.1.2 Performance Constraints

The most important operations that affect performance of the system will be computer vision operations. Because of this fact, we handle these operations at server machine that has powerful CPU than mobile device has. Communication between mobile device and server is achieved with wireless communication protocol. Since project does not involve designing a new hardware and network protocol, the system will be limited by capacity of current hardware implementations.

3.1.3 Portability Constraints

Since the mobile side of the application is developed using Java ME, it can be used most of the mobile devices which support Java ME. Another constraint for mobile device which this system is set up on, the device must have a touchscreen due to overall system is developed considering input type as touchscreen.

Server side of the application is developed using C++ and Matlab, therefore it can be ported any server which provides basic Matlab and C++ support easily.

3.1.4 Hardware Constraints

<u>Server Device</u>

- Server must have an adequate processor for heavy computer vision operations. (At least 2.0 GHz Intel or AMD Processor.)
- Server must be connected to internet.
- Server must have an adequate RAM capacity for computer vision operations. (At least 512 Mb DDR2 Memory.)
- An Apache HTTP server must present on server to provide data transition.

<u>Mobile Device</u>

- Mobile device must have a Wi-Fi connection.
- Mobile device must have a touchscreen.
- Mobile device must have a camera.
- Mobile device must have a color screen with at least 12-bit color resolution (4096 colors).

- Mobile device must support HTTP connection.
- Mobile device must have an adequate RAM capacity for map processing (At least 512 Kb)

3.1.5 Software Constraints

Server Device

- This application can run on all distributions of Linux, MS Windows 95/98, MS Windows NT/2000/XP, 32-bit and 64-bit MS Windows Vista, 32-bit and 64-bit Windows Seven, 32-bit and 64-bit, Windows Server 2003/2008.
- Since Matlab is used for computer vision operations, Matlab must be set up on the server device.
- Apache HTTP Server must be installed and running.

Mobile Device

- This application can run on all operating systems that support Java ME.

3.2 Design Goals and Guidelines

Since mobile devices have not CPU powerful enough, the image processing tasks are handled by the server which is a computer that can handle complicated image processing tasks. This situation changes the design. Speed gain is achieved by this way to increase the performance of the product. However, to do that another hardware is added to the design and wireless communication with the server is required.

To keep the design simple, the data transactions between the server and the mobile device is reduced. Only the data required for image processing tasks and messaging tasks (between the user in the server side and mobile device user) are transacted between the server and mobile device and the other data are kept on the mobile device. By this way, database access is not needed.

Data types coming from the server according to the image processing results are primitive. This also increases performance by reducing the complexity of the data types transferred.

4 Data Design

4.1 Data Description

There is no database in the system. However some necessary storage is done in simple text files. These files include:

- File of messages: This file is stored on the server side. It contains numbered message texts to be sent to the user on the mobile device side. Message class in 5.2.1 in Messaging package is constructed using the message texts in this file.
- File of troops: This file contains the coordinates, types and names of the troops. Troop class in 5.2.3 in Map package is constructed by reading this file.
- Place file: The file includes the names and coordinates of the places. Place class is constructed by reading this file.

Other than the files, the frames captured by the camera of the mobile device are the most important data manipulated. These frames are created in the mobile device and sent to the server to be evaluated. Computer Vision package in section 5.2.5 gets these frames via wireless communication and constructs objects of Frame class using the incoming frames.

4.2 Data Dictionary

The overall system consists of seven packages and each package has several classes in it. Packages, classes in these packages and the data stored in classes are explained in sections below.

4.2.1 Messaging Package

4.2.1.1 Class Message

<u>Attributes:</u>

- String sender: This field keeps the name of the sender of the message object. If the sender does not set this field it is set as "Command Center" as default.
- Date date: This attribute keeps the date and time information.
- String message: The message which is sent to the remote device is kept in this field of the Message object as a string.

Methods:

- void set_sender(String) : this method takes a string as an argument and sets the sender field with this argument.
- String get_sender (): this method returns the sender field, and it does not take any argument.
- void set_date(Date): this method takes a Date object as an argument and sets the date field with this argument.
- Date get_date (): this method returns the date field, and it does not take any argument.
- String get_message (): this method returns the message text, and it does not take any argument.

4.2.1.2 Class MessageController

Attributes:

- File* messages: This is a file which includes all messages. User selects one of these messages via GUI.
- Message currentMessage: this field is set when the user selects one of the messages. The return value of this selection process is currentMessage object.

Methods:

- MessageController (fileName): it is the constructor of the MessageController class. It takes a file name as an argument and initializes the messages attribute.
- void set_currentMessage(Message): it takes a Message object as an argument, and put this object to the currentMessage field.
- Message get_currentMessage (): it returns the currentMessage field of the MessageController class.

4.2.2 GPS Package

4.2.2.1 Class GPSData

Attributes:

- int longitude: longitude information of the place.
- int latitude: latitude information of the place.
- int altitude: altitude information of the place.

- int speed: speed information of the user, it is coming from the Computer Vision package.

Methods:

- GPSData (int longitude, int latitude, int altitude, int speed): it is the constructor of the GPSData class. It initializes the corresponding fields by its arguments.
- void set_longitude(int): It sets the longitude attribute of the object.
- void set_latitude(int): It sets the latitude attribute of the object.
- void set_altitude(int): It sets the altitude attribute of the object.
- void set_speed(int): It sets the speed attribute of the object.
- int get_longitude(): It returns the longitude value of the object.
- int get_latitude(): It returns the latitude value of the object.
- int get_latitude(): It returns the altitude value of the object.
- int get_speed(): It returns the speed value of the object.

4.2.2.2 Class GPSSimulator

Since the mobile device used in this project does not have GPS hardware, GPSSimulator is developed to simulate the GPS part.

Attributes:

- File* GPS_file: it keeps all locations. It is assumed that the user moves on these locations respectively. The time when the user change location is calculated by using the speed of the user.
- GPSData [] data: all locations are taken from the file and they are put into this 'data' array.
- GPSData currentData: the current location of the user.

Methods:

- GPSSimulator (string fileName): It takes a file, in which all locations are listed, as an argument. And this constructor initializes the GPSData array.
- GPSData get_currentData (): it returns the information about the place at where the user is assumed to be.
- void update_currentData(): it changes the place information in the currentData with the next coordination of the user.

4.2.3 Map Package

4.2.3.1 Class Map

Attributes:

- int latitude: it keeps the latitude information
- int longitude: it keeps the longitude information
- char[latitude][longitude][3] RGB: this is a RGB array which has three dimension and keeps the red, green, and blue intensity values of each pixel.
- int zoom_level:it keeps the information about the zooming. The details on the map are changed according to this zoom_level and this field changes according to the movement of the user.
- Place[] places: it keeps all Place objects which can be shown or hidden on the map according to the user request and the environmental condition.
- Troop[] troops: it keeps all Troop objects, they can be shown or hidden according to the user request and the environmental changes.

Methods:

- void set_zoom_level(int): this method takes an integer as an argument and it sets the zoom_level attribute.
- void adjust_RGB_Array(char[latitude][longitude][3]): this function takes an rgb array, and it changes the RGB attribute according to this argument.

4.2.3.2 Class Place

Attributes:

- int latitude: it keeps the latitude of the Place object.
- int longitude: it keeps the longitude of the Place object.
- int altitude: it keeps the altitude of the Place object
- boolean isVisible: it determine that the place information is going to be visible or not.
- double distance: it is the distance between the place where user wants to see information about, and he current location of the user.

Methods:

- void set_visible(): it changes the value of the isVisible. If it is already true, it makes it false or if it is already false, it makes it true.

4.2.3.3 Class Troop

Attributes:

- int latitude: it keeps the latitude information of the place at which the troop is.
- int longitude: it keeps the longitude information of the place at which the troop is.
- int size: it keeps the number of the people in the troop.
- string name: the name of the troop is kept in this field.
- string type: it specifies that the troop is air force, army force or marine forces.
- boolean is_visible: it is set to one if troops are shown, it is set to zero if they are not shown.
- boolean is_enemy: it is to decide the troop is enemy troop or ally troop.

Methods:

- void setVisible(): this function does not take an argument, because, it negates the value in the isVisible field.

4.2.4 Camera Package

4.2.4.1 Class CameraController

This class is just responsible for capturing image and it has no attribute.

4.2.5 Computer Vision Package

4.2.5.1 Class ComputerVision

Attributes:

- int velocity: it is one of the return values of the image processing task, it is set with the velocity information extracting from snapshots.
- int luminosity: it is the other return value of the image processing task, it is set with the environmental light condition information extracting from the snapshots.
- Frame[] frame: it keeps all frames (snapshots) coming from the mobile device. It sends them to the evaluate_frames function one by one.

Methods:

- void evaluate_frames(): it takes frames from the 'frame' attribute of itself and process them and set the velocity and luminosity fields.
- void update_frame_buffer(Frame[]): it updates the frame buffer with the new frames coming from the remote mobile device via wireless.

4.2.5.2 Class Frame

<u>Attributes:</u>

- int height: it keeps the height value of the snapshot (frame).
- int weight: it keeps the width value of the snapshot (frame).
- char[height][width][3] rgb_values: it is an array of char type, it has three dimension. Height and width is to keep all pixels and 3 are to keep red, green, and blue intensity values of each pixel.

<u>Methods:</u>

- Frame (int height, int width, char rgb[][][]): it is the constructor of the Frame class. It creates a new instance whose attributes are height, width and rgb [][][].
- void set_height(int): it sets the height attribute of the object.
- int get_height(): it returns the value of the height attribute.
- void set_width(int): it sets the width attribute of the object.
- int get_width(): it returns the value of the width field.
- void set_rgb_values(char[][][]): it sets the rgb array of the object.
- char[][][] get_rgb_values(): it returns the rgb array of the object

4.2.6 Screen Package

4.2.6.1 Class Screen

<u>Attributes:</u>

- int brightness: it is the value for the brightness of the screen.
- int contrast: it is the contrast value of the screen.
- Color color: background color of the screen.
- Map map: the map part which is shown. The map part is decided according to the user's current position.
- InformationBar info_bar: this InformationBar object is used to show information about the place whose information is needed by the user.
- ButtonBar button_bar: The button_bar is a ButtonBar object and it keeps all buttons on it.
- MessageBox message_box: A message is shown on the screen if any message is sent from the server side. This object keeps that message.

Methods:

- void set_brightness(int): it sets its argument to the brightness field.
- void set_contrast(int): it sets its argument to the contrast field.
- void set_color(Color): it sets its argument to the Color object in the class
- int get_brightness(): It returns the brightness value of the object.
- int get_contrast(): It returns the contrast value of the object.
- Color get_color (): It returns the Color value of the object.
- InformationBar get_infoBar (); It returns the infoBar field of the object.
- ButtonBar get_buttonBar (); It returns the buttonBar field of the object.
- MessageBox get_messageBox (): It returns the messageBox value of the object.
- int get_contrast(): It returns the contrast of the object.

4.2.6.2 Class MessageBox

Attributes:

- int fontSize: it specifies the font size of the message when it is shown on the screen. It changes according to the environmental changes.
- String message: It keeps the text which is going to be shown on the screen.
- boolean isVisible: it is a boolean which decides the message text is shown or not.
- Color color: It is the color of the text. This field also changes while environmental luminosity changes.

Methods:

- void set_fontSize(int): It update the fontSize field which changes the font size of the messages shown on the screen.
- int get_fontSize(): It returns the fontSize value as a return parameter.
- void set_message(String): It sets the message attribute to a string which is going to be shown on the screen.
- String get_message (): It returns the message text as a string.
- void set_visible(bool): it arranges the visibility of the MessageBox object.
- bool get_visible(): It returns the boolean value inside the isVisible attribute.
- void set_color(Color): It sets the color field by a Color object.
- Color get_color (): It returns the color field.

4.2.6.3 Class ButtonBar

Attributes:

- int size: the size of the buttons is important for context aware user interface design. Buttons are scaled according to this size information.
- boolean isVisible: it specifies the visibility of the bar. If isVisible is true then the button bar is drawn on the screen. If it is false, the button bar is not shown.
- Button [] buttons: it is an array of Button objects which are shown if the ButtonBar is opened.
- Color color: It is the color of the ButtonBar.

Methods:

- void set_size(int): It sets the size field with its argument.
- int get_size(): It returns the size of the Button object.
- void set_visible(bool): It sets the visibility of the object.
- bool get_visible(): It returns a boolean which represents the object is visible or not.
- void add_button(Button): It adds the given Button object to the buttons field which is an array of Button objects.
- Button get_button (int id): It returns the Button object whose index in the array is equal to the argument of the function, id.
- void set_color(Color): It sets the color of the object.
- Color get_color (): It returns the color field.

4.2.6.4 Button

Attributes:

- int size: It is the size of the button. It changes according to the user movement.
- boolean isVisible: it is used when drawing buttons on the screen. If isVisible attribute of an object is true, then this button is drawn, if it is false, then the button is not drawn on the screen.
- int priority: Each Button object has a priority value. This priority helps the application decide which buttons are shown on the screen and which ones are listed under one button when environment condition changes.
- Color color: It is the color of the button.

Methods:

- void set_size(int): It sets the size field to its integer argument.
- int get_size(): It returns the value of the size field.
- void set_visible(bool): It changes the visibility of the object.
- bool get_visible(): It returns the information about the buttons are visible or not.
- void set_priority(int): It sets the priority field to the its argument.
- int get_priority(): It return the priority of the object.
- void set_color(Color color): It sets the color of the buttons.
- Color get_color (): It returns the color of the button.

4.2.6.5 InformationBar

Attributes:

- int fontSize: it is the font size of the text which is shown.
- string information: it is the string which is going to be shown on the InformationBar.
- int zoomLevel: it is also shown in the InformationBar. It gives the user information about the zoom-level of the map.
- boolean isVisible: if this field is true, the information bar is shown and if it is false, the bar is not shown on the screen.
- Color color: it is the color of the information bar.

Methods:

- void set_fontSize(int): It sets the fontSize of the InformationBar object.
- int get_fontSize(): It returns the fontSize of the object.
- void set_information(String):
- String get_information():

4.2.7 GUIManager Package

4.2.7.1 Class GUIManager

Attributes:

- contextMngr: it is a ContextManager object. It is used to access methods in ContextManager class and to get information coming from the ComputerVision component via ContextManager class.

- interactionMngr: this is an InteractionManager object; it is used to access methods in UserInteractionManager and to get information coming from the User via UserInteractionManager class.

Methods:

- void configureButtons(): It configures the buttons on the button bar. It increment the size of the buttons and it only draws the buttons which have higher priorities, other buttons listed under one button, the user can access them if he/she wants. It does not take an argument, it uses contextMngr and interactionMngr objects to decide the buttons. Also, the colors of the buttons are configured.
- void configureInfoBar(): It configures the information bar. It increments the font size of the text if the user in motion. If the information bar is not used for a while, it hides the bar.
- void configureVisibility(): It decides the visibility constraints.
- void configureMessage(): If the user is in motion, this function just takes the important parts of the message which is written in the file between *** and ***. It decides the text which is going to be shown and the color of the text.
- void configureMap(): It configures the Map object by changing its colors to sustain the visibility, and the details on the map to sustain the usability. It can change the current colors to the more visible ones based on the environment condition. It also reduces the details on the map. It can show or hide the enemy and ally troops, name of the places according to the user's movement.

4.2.7.2 Class ContextManager

Attributes:

- int velocity: this field keeps the information about the user's velocity.
- int luminosity: this attribute has the information about the environment luminosity.
- int threshold_velocity: it is a threshold value for the velocity. It is used when the user interface changes according to the user's movement.
- int threshold_luminosity: it is a threshold value for the luminosity. It is used when the colors of the user interface changes according to the environment luminosity.

Methods:

- void configureButtons(): It configures the buttons on the button bar. It increment the size of the buttons and it only draws the buttons which have higher priorities, other buttons listed under one button, the user can access them if he/she wants. It does not take an argument, it uses contextMngr and interactionMngr objects to decide the buttons. Also, the colors of the buttons are configured.
- void configureInfoBar(): It configures the information bar. It increments the font size of the text if the user in motion. If the information bar is not used for a while, it hides the bar.
- void configureVisibility(): It decides the visibility constraints.
- void configureMessage(): If the user is in motion, this function just takes the important parts of the message which is written in the file between *** and ***. It decides the text which is going to be shown and the color of the text.
- void configureMap(): It configures the Map object by changing its colors to sustain the visibility, and the details on the map to sustain the usability. It can change the current colors to the more visible ones based on the environment condition. It also reduces the details on the map. It can show or hide the enemy and ally troops, name of the places according to the user's movement.

4.2.7.3 Class UserInteractionManager

If the user wants to arrange some attributes manually, his/her requests are delivered to the GUIManager class via this UserInteractionManager class. (i.e., when the user is not in motion, he/she may not like the zoom-level of the map and he/she may want to adjust this property.)

Attributes:

- int preferred_zoomLevel: it is the requested zoom level by the user.
- int preferred_brightness: it is the requested brightness value by the user.
- int preferred_contrast: it is the requested contrast value by the user.
- boolean wantMessageBox: if the user wants to see the message box again, he/she has to request this from the device manually. This field holds the user's request.

- boolean wantInfoBar: if the user wants to see an information about a place, he/she requests this from the device manually. This field holds the user's request.
- boolean wantButtonBar: Buttons are hidden as default. This field holds the user's request about whether he/she wants to use buttons or not.
- boolean wantEnemies: it holds the information about whether the user wants to see enemies or not.
- boolean wantAllies: it holds the information about whether the user wants to see allies or not.
- boolean wantPlaceNames: it holds the information about whether the user wants to see place names on the map or not.

Methods:

- void zoomIn/Out(int): It takes the zoom-level from the user and send this request and the data to the GUIManager.
- void changeBrightness(int): It takes the brightness value from the user and deliver this request and the data to the GUIManager.
- void changeContrast(int): It takes the contrast value from the user and give it to the GUIManager to adjust the screen according to this information.
- void showMessage(bool): If the user wants to see an old message, this function takes his/her request and deliver it to the GUIManager.
- void showInformation(bool): If the user needs to see a place information which is not seen on the screen because of the user's movement, his/her request is sent to the GUIManager by this method.
- void showButtonBar(bool): If the user wants to use buttons, he/she needs to open the button bar. His/her request is sent to GUIManager by this method.
- void showEnemies(bool): If the user wants to see the place of the enemy troops, this is provided by the GUIManager and it needs to communicate with this method to be able to notice the request.
- void showAllies(bool): If the user wants to see the place of the ally troops, this is provided by the GUIManager and it needs to communicate with this method to be able to notice the request.
- void showPlaceNames(bool): the user can show or hide the place names. Actually, this information is shown as default, but according to the user's

movement, they can be hidden by the contextManager. This method takes the user's request. And send it to the GUIManager.

- int get_preferred_zoomLevel():It returns the zoomLevel which the user wants.
- int get_preferred_brightness(): It returns the brightness value which the user wants.
- int get_preferred_contrast(): It returns the contrast value which the user wants.
- bool isMessageWanted(): It returns a boolean which reveal whether the user wants to see the messages or not.
- bool isInfoWanted():It returns a boolean which reveal whether the user wants to see an information about a place or not.
- bool isButtonsWanted(): It returns a boolean which reveal whether the user wants to see buttons bar or not.
- bool isEnemiesWanted(): It returns a boolean which reveal whether the user wants to see the enemy troops or not.
- bool isAlliesWanted(): It returns a boolean which reveal whether the user wants to see the ally troops or not.
- bool isPlaceNamesWanted(): It returns a boolean which reveal whether the user wants to see the names of the places or not.

These properties are arranged by the GUI Manager according to the information extracted from the snapshots by the Computer Vision component. This component just provides the user extra power. If the user is not satisfied with the revised user interface, he/she can adjust it.

4.2.8 Wireless Package

4.2.8.1 Class Wireless Communication

Attributes:

This class is just an interface to send and get data packages. It has no attribute.

Methods:

- bool send_data(TCP Package): this method is actually implemented in classes which wants to send package; because; Wireless is an interface.
- void get_data(TCP Package): this method is actually implemented in classes which wants to get package; because; Wireless is an interface.

5 System Architecture

A description of the program architecture is presented here.

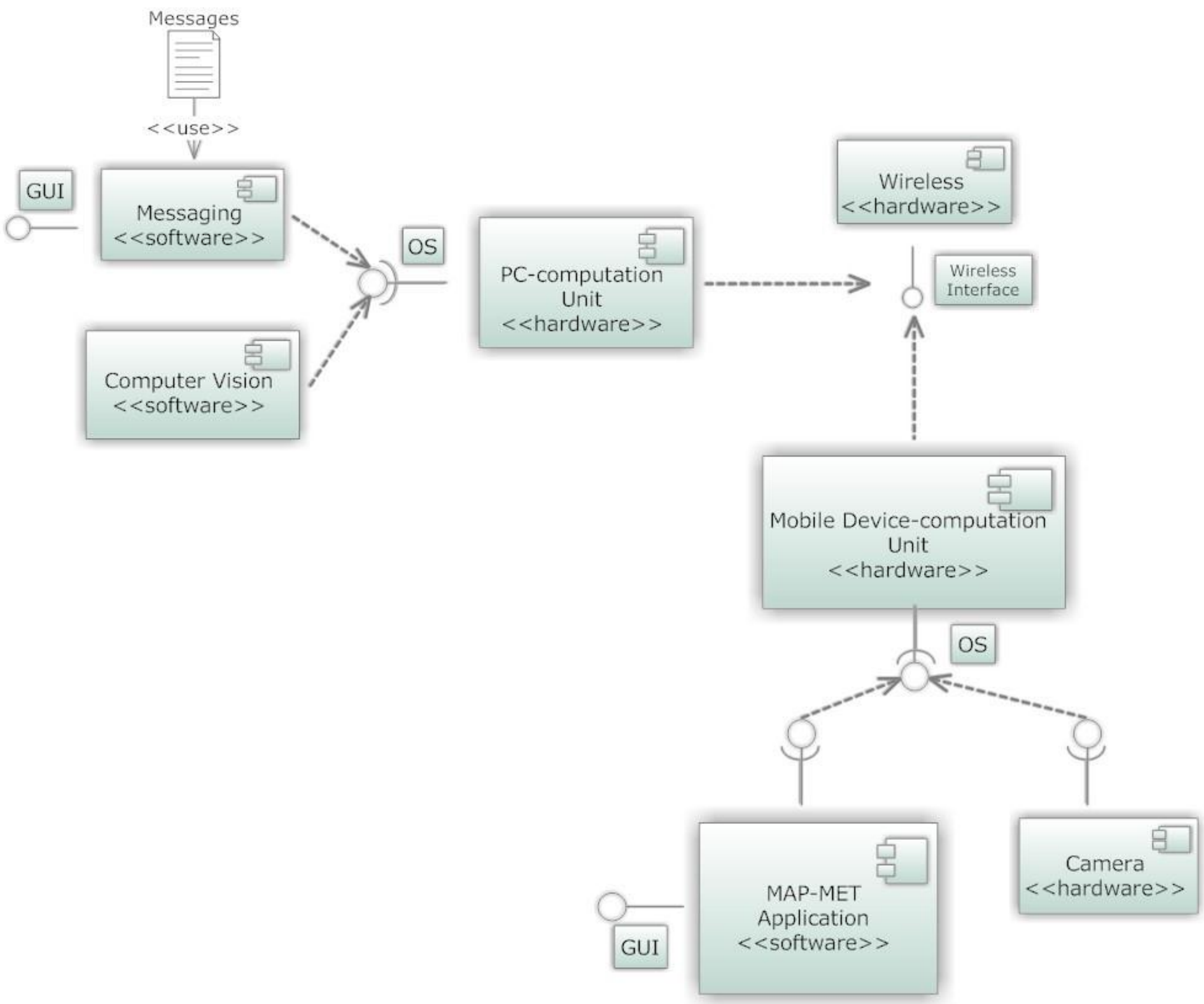
5.1 Architectural Design

MAP-MET has seven main components namely Messaging, Computer Vision, PC-Computation Unit, Wireless, Mobile Device-Computation Unit, Camera and MAP-MET Application.

After the application is started, firstly, the camera of the mobile device takes snapshots continuously. These snapshots are sent to the server side from Mobile Device-Computation Unit via Wireless component. PC-Computation Unit delivers them to the Computer Vision component.

In the server side, Computer Vision component processes snapshots and extracts movement and luminosity information. After extracting this information, they are wrapped in the server side and they are sent from PC-Computation Unit via Wireless component. After this package is received by Mobile Device-Computation Unit and it is unwrapped, it is delivered to the MAP-MET Application component. This component decides the user interface properties such as fonts of the texts, size of the buttons, zoom level of the map, context of the screen according to the movement information and it adjusts colors on the screen according to the luminosity information. The details about how MAP-MET Application and other components work will be explained in Detailed Design Report.

Messaging component has message objects in it, wraps the message chosen by the sender and sends the package to the mobile device via Wireless component. After unpacking process, the message is shown on the screen by MAP-MET Application component.



Messages
<<use>>
GUI
Messaging <<software>>
OS
PC-computation Unit <<hardware>>
Wireless <<hardware>>
Wireless Interface
Computer Vision <<software>>
Mobile Device-computation Unit <<hardware>>
OS
MAP-MET Application <<software>>
Camera <<hardware>>
GUI

Figure 5.1

5.2 Description of Components

The package diagram included below intends to show object-oriented nature of the basic software components of the whole MAP-MET System. The project has two main packages, namely, Server package and Client package. The Server package includes Messaging, Computer Vision and Wireless Communication packages. In the Client package, there are Wireless Communication, GUI Manager, Camera, Screen, GPS and Map packages.

Since implementation of the project will be mainly done using C++ and Java Micro Edition programming languages, software packages are arranged suitably with Object-Oriented way of coding. All packages import the necessary classes and packages as indicated by the diagram.

Server and Client packages represent the main agents of the project. They both communicate with each other, using Wireless Communication packages of their own.

Messaging and Computer Vision packages which are in Server package import from Wireless Communication package since they both use methods of Wireless Communication package to send/get information to/from Client package.

GUI Manager, Screen and Camera packages which exist in Client package import from Wireless Communication package since they all use methods of Wireless Communication package to send/get information to/from Server package.

GUI Manager Package imports from Screen package because it refreshes screen according to user adjustments and information coming from Server.

Screen package imports from Map package since there is always a map shown on the screen. To be able to manipulate map, it needs to access the methods and attributes of the Map package.

Map package imports from GPS package since user location is shown on the map according to GPS value. Moreover, zooming operation is done based on user location.

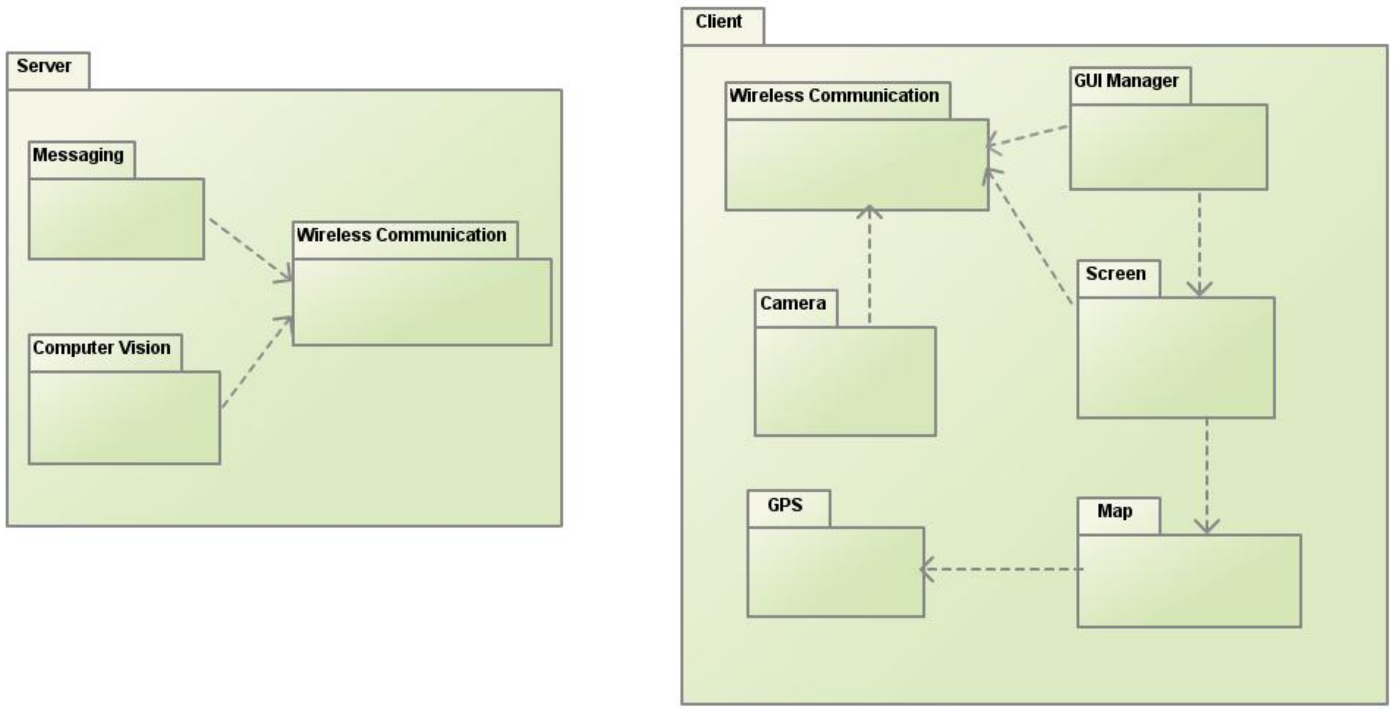


Figure 5.2

5.2.1 Messaging

Diagram for Messaging Component:

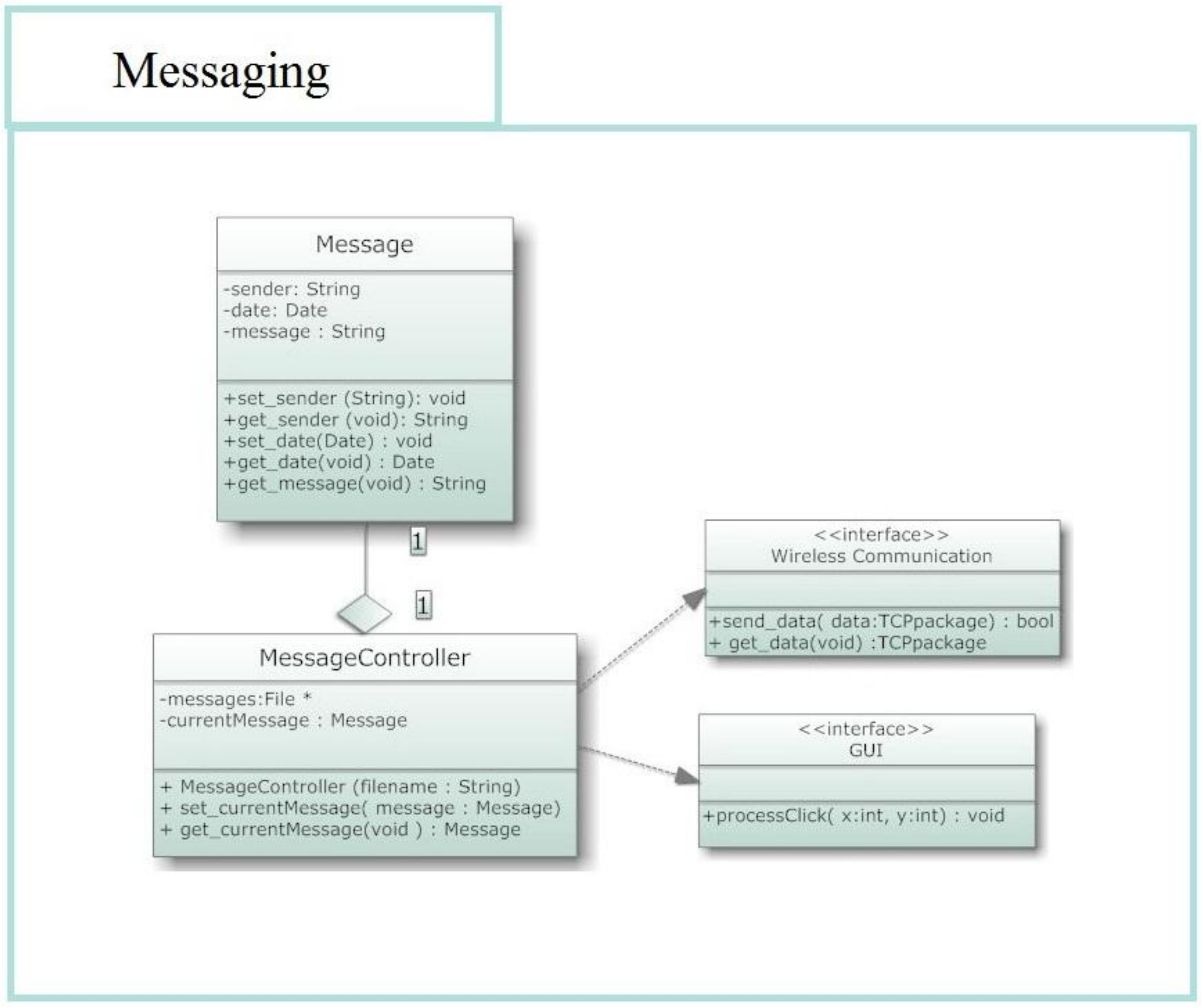

Messaging
Message
-sender: String
-date: Date
-message : String
+set_sender (String): void
+get_sender (void): String
+set_date(Date) : void
+get_date(void) : Date
+get_message(void) : String
1
1
MessageController
-messages:File *
-currentMessage : Message
+ MessageController (filename : String)
+ set_currentMessage(message : Message)
+ get_currentMessage(void) : Message
<<interface>>
Wireless Communication
+send_data(data:TCPpackage) : bool
+ get_data(void) :TCPpackage
<<interface>>
GUI
+processClick(x:int, y:int) : void

Figure 5.3

5.2.1.1Processing narrative for component GPS

This package provides sending message from command center (server) to a remote mobile device (client). On the server side, all messages are kept in a file. Firstly, this package wants server side user to select a message via graphical user interface. The user has an option to arrange sender field. Before sending the message, date and time information is also attached. After that, the message is wrapped and the Messaging Package sends it via Wireless Communication Interface.

5.2.1.2 Interface Description for component Messaging

There are two types of input for Messaging package. First input is a file keeping all messages and the second one is the information about which message is selected by the server side user. If the user wants to set the sender field, name of the sender can be another input for the Messaging component. The only output of this component is a packed message object.

5.2.1.3 Processing Detail for component Messaging

After the user opens this messaging application, the MessageController class takes a file that keeps messages as an input and lists all of these messages on the screen. The user needs to select a message from this list. After the user select a message, MessageController gets the message corresponding to the number of the message selected by the user from file and creates a message object. The default sender of the message is defined as “Command Center” unless the user specifies it manually. When the user clicks on the send button, the date and time information is added and the created message object is wrapped. Finally, the package is sent to the remote device via Wireless Communication component.

5.2.1.4 Dynamic Behavior for component Messaging

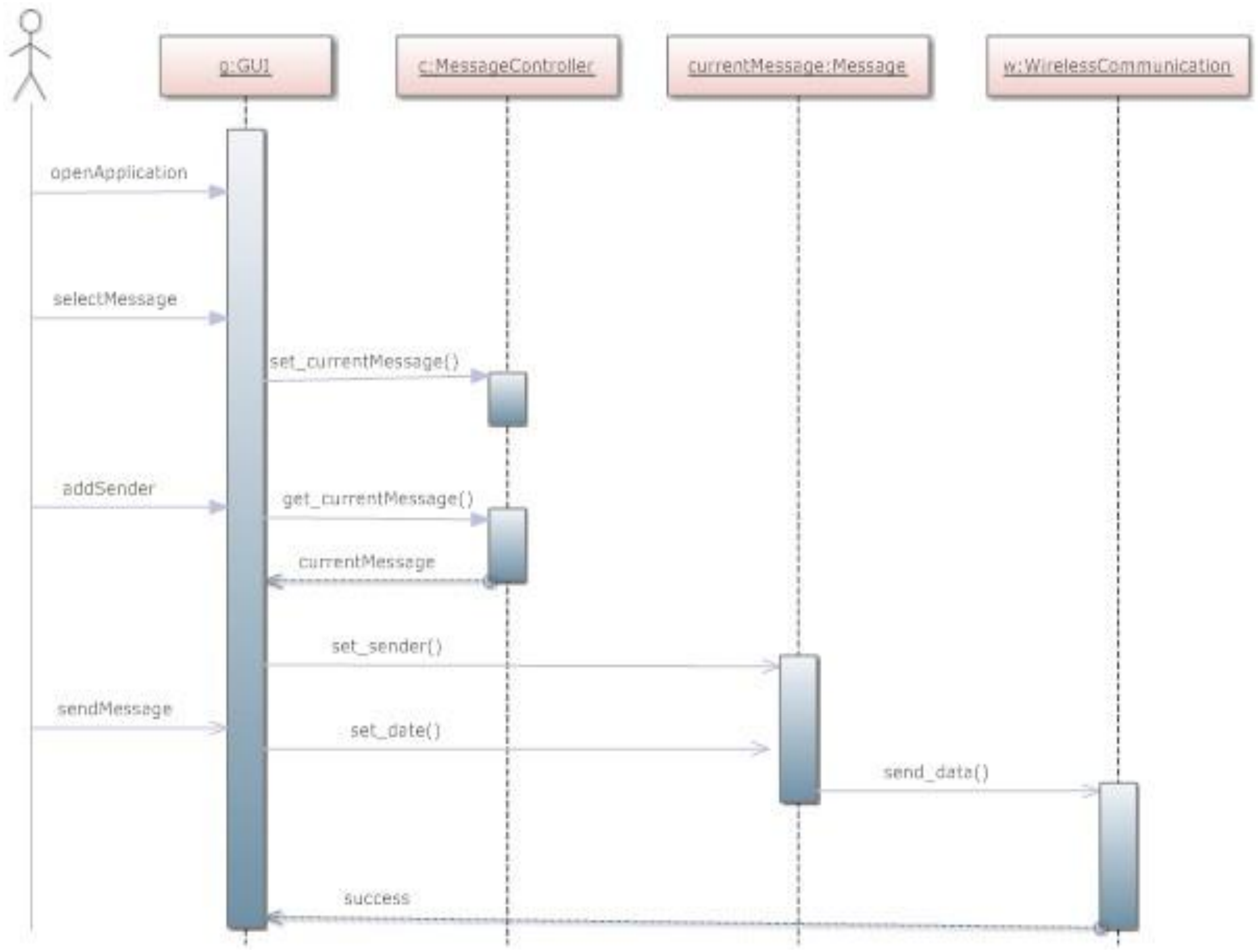


Figure 5.4

5.2.2 GPS

Diagram for the component GPS

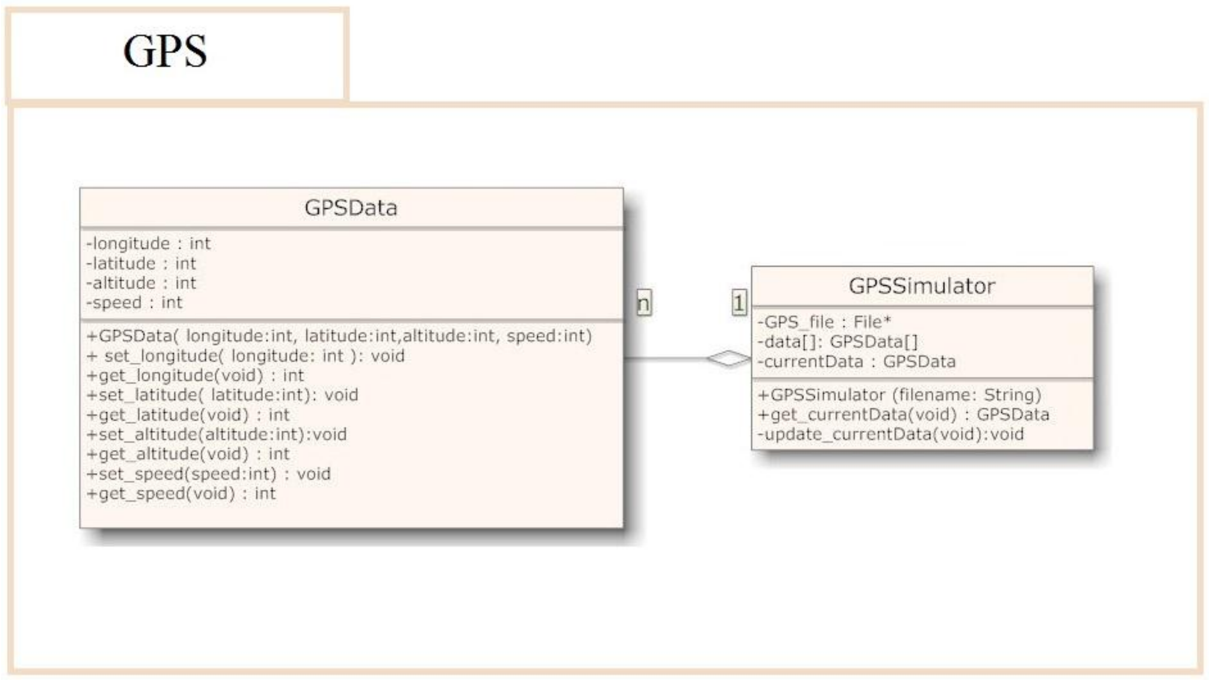

GPS
GPSData
-longitude : int
-latitude : int
-altitude : int
-speed : int
+GPSData(longitude:int, latitude:int,altitude:int, speed:int)
+ set_longitude(longitude: int): void
+get_longitude(void) : int
+set_latitude(latitude:int): void
+get_latitude(void) : int
+set_altitude(altitude:int):void
+get_altitude(void) : int
+set_speed(speed:int) : void
+get_speed(void) : int
n
1
GPSSimulator
-GPS_file : File*
-data[]: GPSData[]
-currentData : GPSData
+GPSSimulator (filename: String)
+get_currentData(void) : GPSData
-update_currentData(void):void

Figure 5.5

5.2.2.1 Processing narrative for component GPS

This component is used for finding the location of the user. This information is necessary for the GUI Manager component. It refreshes the map by taking the place of the user as the center. In other word, when the zoom level of the map is changing or when the application starts, the user coordinate must be the center. In addition, to calculate the distance between the user and the enemy or ally troops, this information is needed again.

5.2.2.2 Interface Description for component GPS

Since there is not GPS hardware on the mobile device used in the project, GPS is simulated by this component. This component has two classes, namely, GPSData and GPSSimulator. There is only one input which GPS component takes from the other components. This input is speed information of the user and it is taken from the Computer Vision component. It is used to simulate the GPS hardware. Three outputs of the GPS component are longitude, latitude and altitude information of the place where the remote user is and these outputs are sent to the GUI Manager component.

5.2.2.3 Processing Detail for component GPS

After the user starts the application, before displaying the screen content, the first place information in the GPSData file in the GPSSimulator class is taken and it is assumed that the user is at that place. After that, the speed information is taken. If speed information is not zero, the division of the distance between the user's coordinates and the next coordinate in the file by speed gives the time that the user's coordinates changes. When these tasks are done, GUI Manager Component takes the new location and assumes that the user is at this place.

5.2.2.4 Dynamic Behavior for component GPS

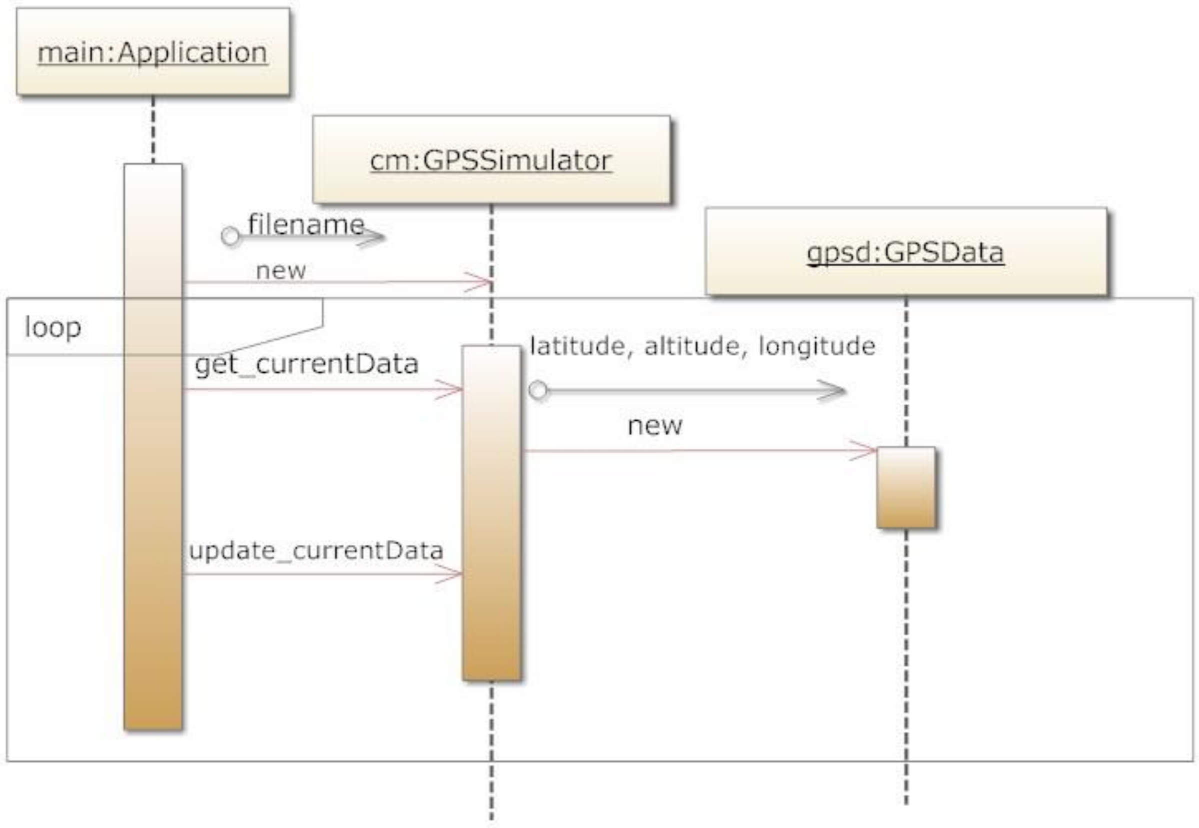


Figure 5.6

5.2.3 Map

Diagram for the component Map

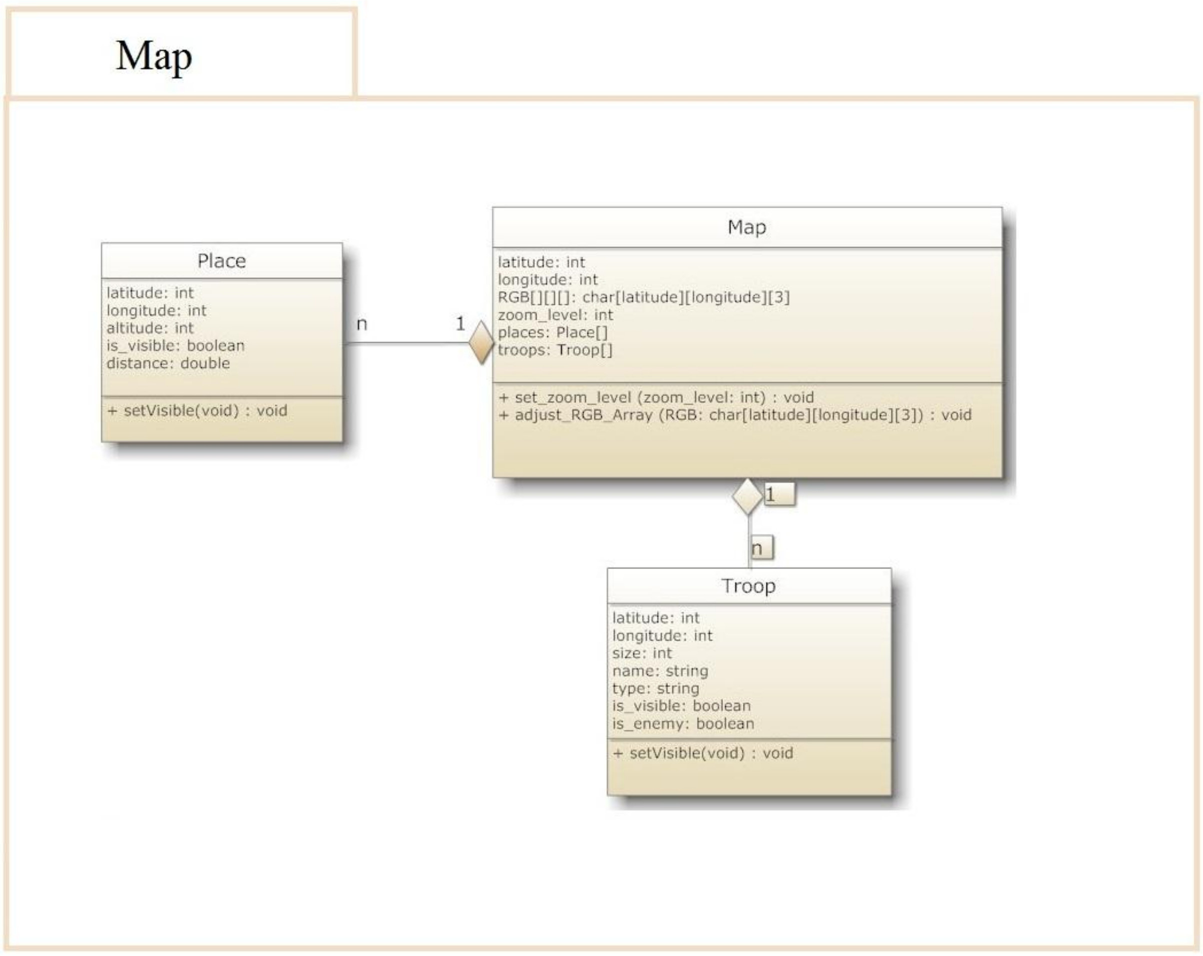


Figure 5.7

5.2.3.1 Processing narrative for component Map

This component deals with the map drawn on the screen. When the user starts application, he/she sees a map whose center is the location of the user. This map component has to have information about places and the ally and enemy troops because if the user wants to get information about a place or to see the places of other troops, the map must provide this information. In addition, as the light condition changes, the Map component adjusts the color on the map according to the information sent by the GUI Manager component.

5.2.3.2 Interface Description for component Map

There are three classes in Map component which are Place class, Troop class, and Map class. This component takes five inputs from other components. One of the inputs is the information about the colors which needs to be change according to the environmental changes. This input is an RGB array, and it includes the revised RGB values of the map. The second input is a Boolean, and it is coming from GUI Manager to specify whether the other troops need to be shown on the screen or not. The third input, zoom_level, is also from GUI Manager, and specifies the zoom-level of the map. The other input is the location information of the user which is sent by GPS component. Finally, the last input is again coming from the GUI Manager component, and it marks the places in the Map component for making them visible or invisible. The output of the Map component is a Map object which is revised and it is sent to the Screen component.

5.2.3.3 Processing Detail for component GPS

The Map component needs the location information of the user before it begins to create the Map object which is sent to the Screen class. After GPS component send this information, Map class takes the necessary information of the user interface properties from GUI Manager Component. These are visibility of the places and troops, zoom-level of the map and an RGB array for color changes. If the user is not in motion, GUI Manager does not continuously send visibility and zoom-level information and this is also true for the RGB array in a situation in which light condition is not changing. If the user is in motion or the luminosity changes, after these changes are set, a new Map object is created and it is sent to the Screen component by the Map component.

5.2.4 Camera

<u>Diagram for the component Camera</u>

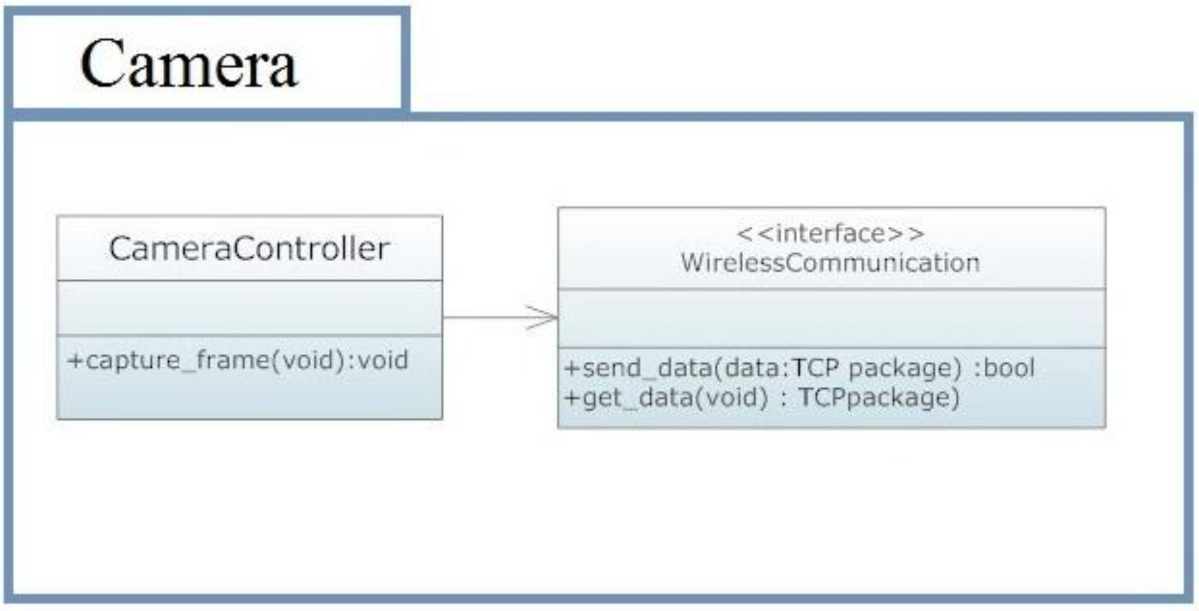

Camera
CameraController
+capture_frame(void):void
<<interface>>
WirelessCommunication
+send_data(data:TCP package) :bool
+get_data(void) : TCPpackage)

Figure 5.8

5.2.4.1 Processing narrative for component Camera

This package works to communicate with the camera hardware of the mobile device and tells it to capture frames. These frames are sent to server with WirelessCommunication interface.

5.2.4.2 Interface Description for component Camera

The only output of this component is a package consisting of frames captured by the camera of the mobile device.

5.2.4.3 Processing Detail for component Camera

This component is active until the application on the mobile device is closed. This means that the component is actively working all the time. The reason is that to change the user interface dynamically, these frames have to be sent to server and evaluated there.

The captured frames are packed and sent to server to be evaluated via WirelessCommunication interface.

5.2.4.4 Dynamic Behavior for component Camera

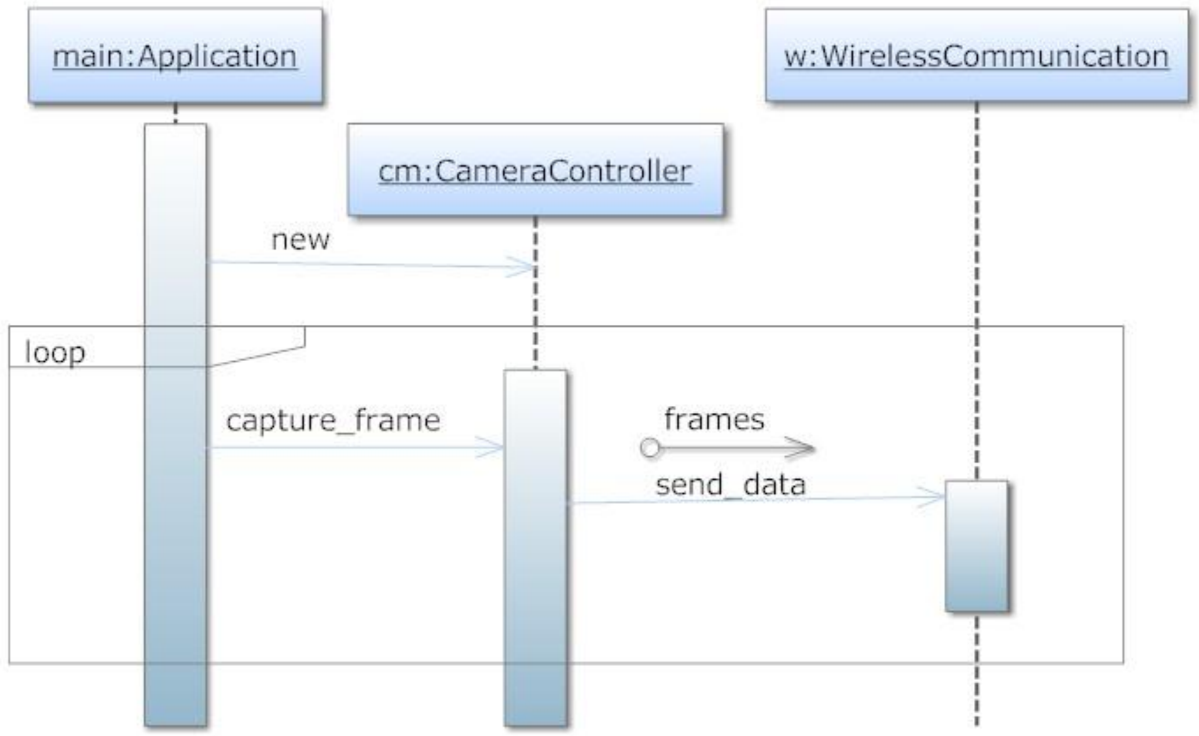


Figure 5.9

5.2.5 Computer Vision

Diagram for the component Computer Vision

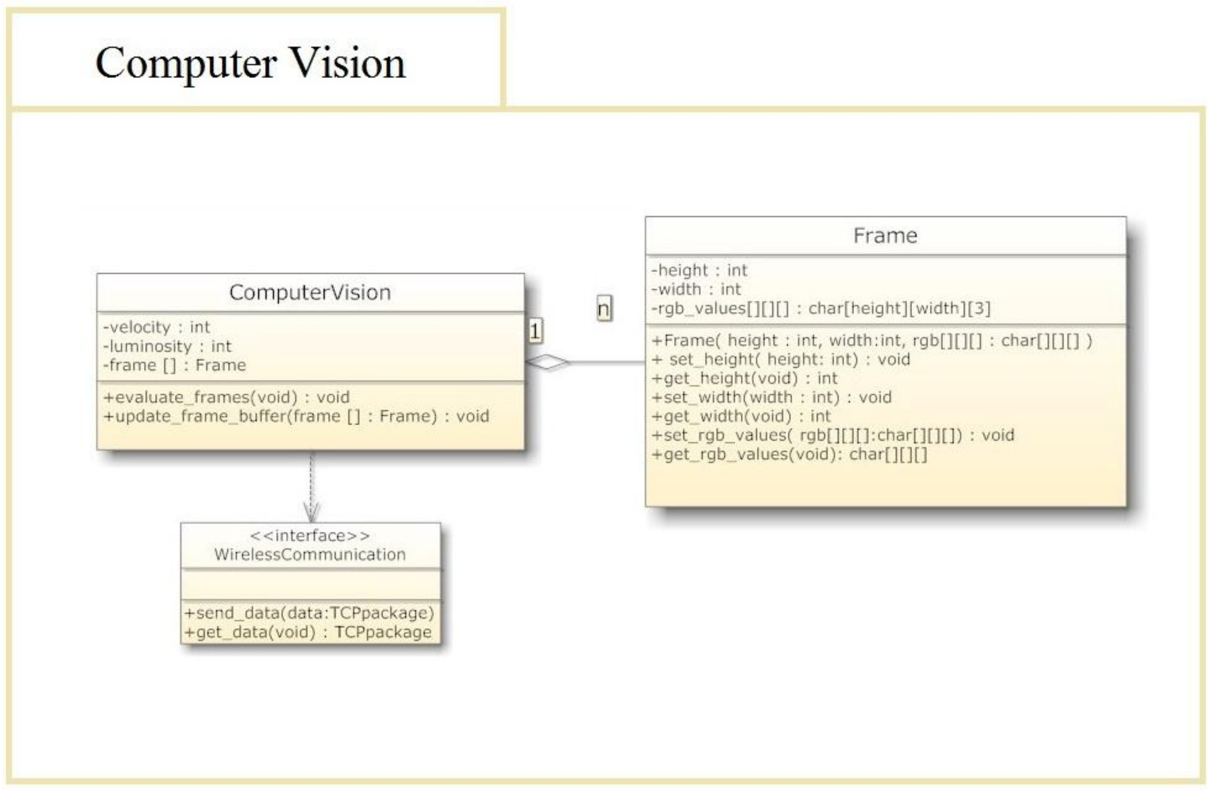

Computer Vision
ComputerVision
-velocity : int
-luminosity : int
-frame [] : Frame
+evaluate_frames(void) : void
+update_frame_buffer(frame [] : Frame) : void
1
n
Frame
-height : int
-width : int
-rgb_values[][][] : char[height][width][3]
+Frame(height : int, width:int, rgb[][][] : char[][][])
+ set_height(height: int) : void
+get_height(void) : int
+set_width(width : int) : void
+get_width(void) : int
+set_rgb_values(rgb[][][]:char[][][]) : void
+get_rgb_values(void): char[][][]
<<interface>>
WirelessCommunication
+send_data(data:TCPpackage)
+get_data(void) : TCPpackage

Figure 5.10

5.2.5.1 Processing narrative for component Computer Vision

This component works on server side to evaluate the frames coming from mobile device via WirelessCommunication interface. When the application starts, this component starts to work. It determines the velocity of the user and the quantity of lighting of the environment the user is in. After calculating these values, this component sends these values to the mobile device.

5.2.5.2 Interface Description for component Computer Vision

The inputs for this component are the packed frames sent by the mobile device. These frames are captured with the camera of the mobile device. Outputs of this component are the calculated velocity and luminosity values to be used by the mobile device in order to dynamically change the user interface.

5.2.5.3 Processing Detail for component Computer Vision

The packed frames come from the mobile device via WirelessCommunication interface. ComputerVision class updates the frame array consisting of frame objects according to the new frames with its update_frame_buffer function. New frame objects are constructed using the constructor of the Frame class.

ComputerVision class traverses the frame array (frame []) and by comparing consecutive frame objects in the array by evaluate_frames function it sets the luminosity and velocity values. Then, these calculated values are sent to the mobile device with the help of WirelessCommunication interface.

5.2.5.4 Dynamic Behavior for component Computer Vision

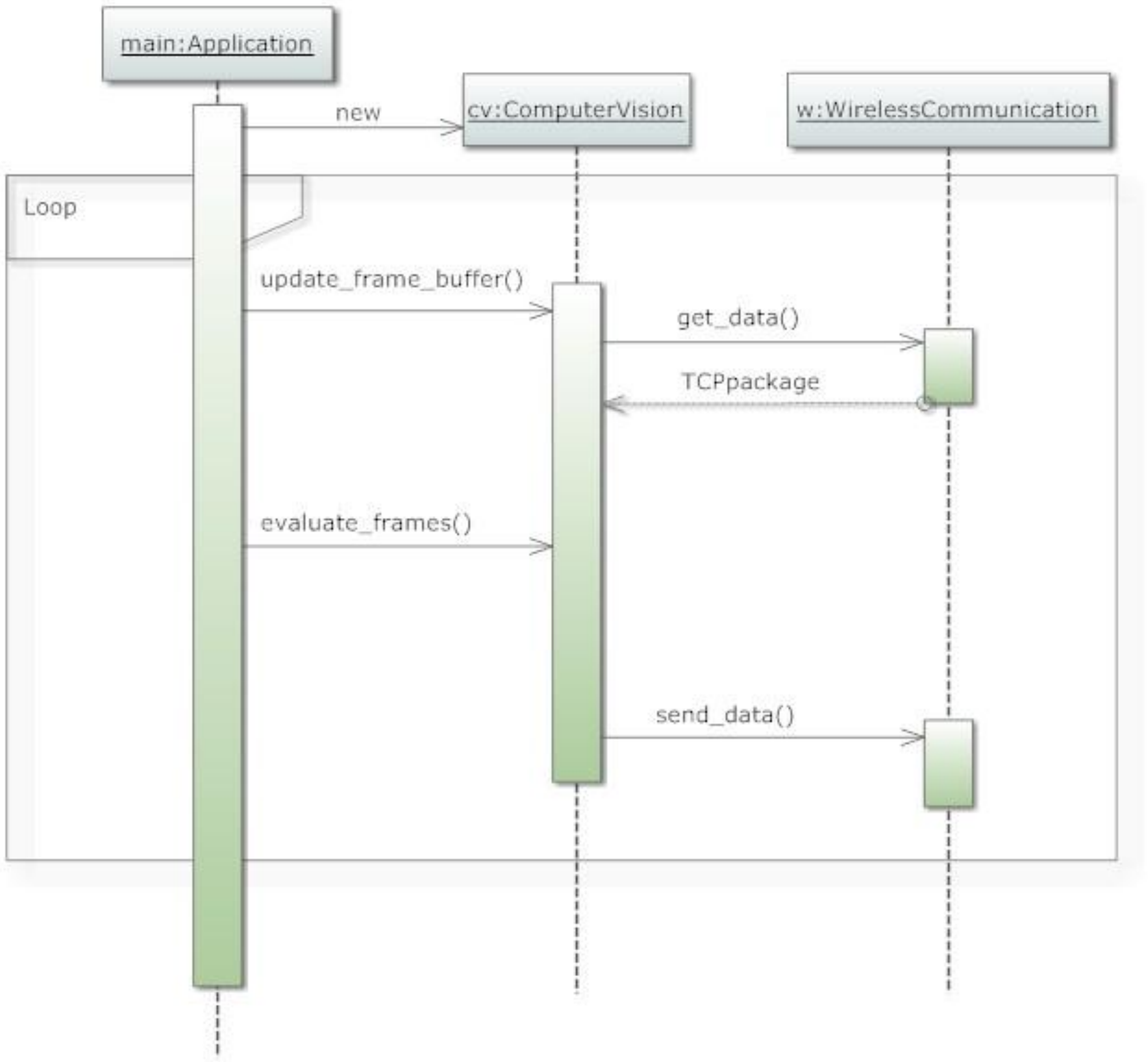


Figure 5.11

5.2.6 Screen

<u>Diagram for the component Screen</u>

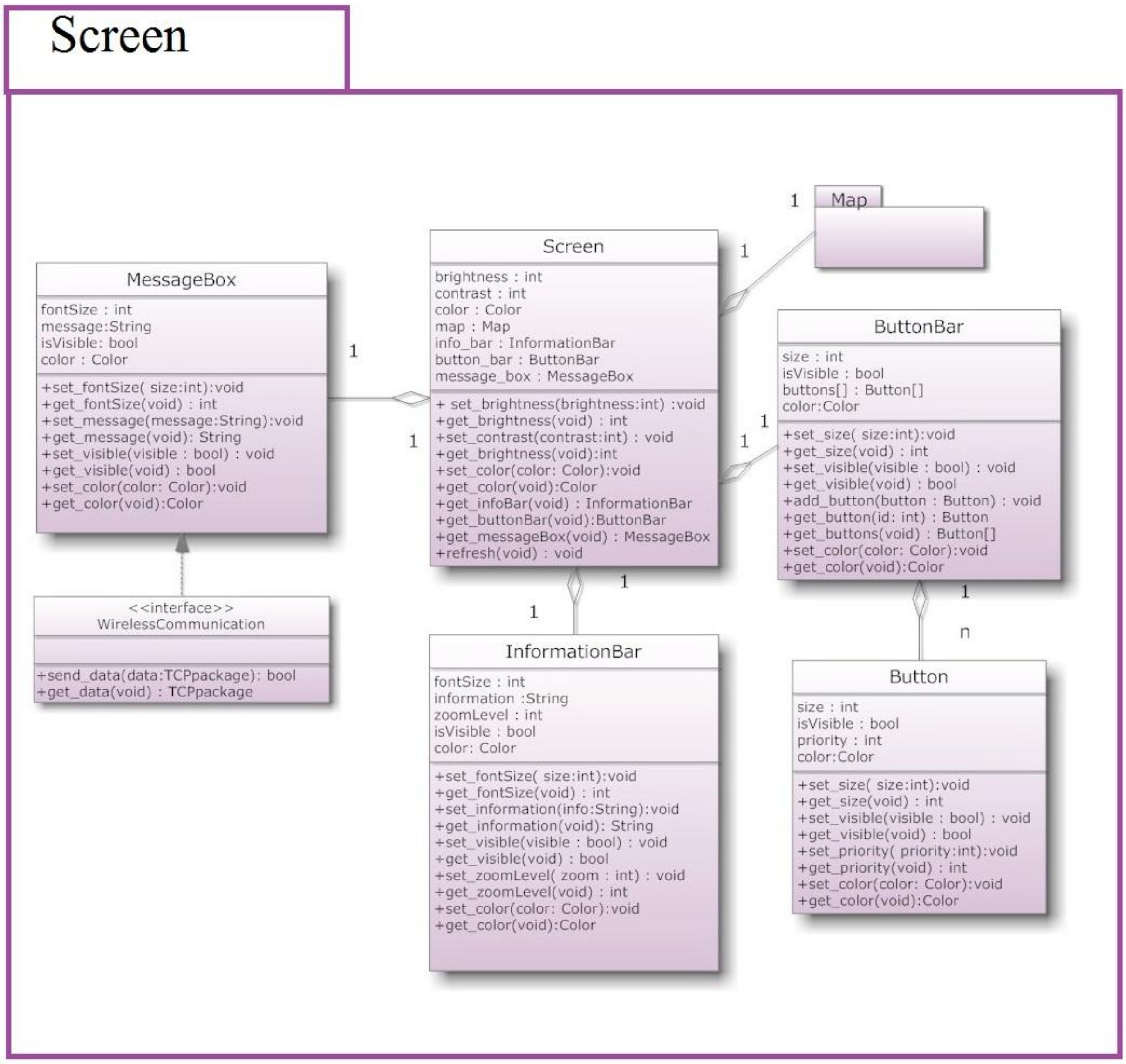

Screen
Map
1
Screen
brightness : int
contrast : int
color : Color
map : Map
info_bar : InformationBar
button_bar : ButtonBar
message_box : MessageBox
+ set_brightness(brightness:int) :void
+get_brightness(void) : int
+set_contrast(contrast:int) : void
+get_brightness(void):int
+set_color(color: Color):void
+get_color(void):Color
+get_infoBar(void) : InformationBar
+get_buttonBar(void):ButtonBar
+get_messageBox(void) : MessageBox
+refresh(void) : void
1
MessageBox
fontSize : int
message:String
isVisible: bool
color : Color
+set_fontSize(size:int):void
+get_fontSize(void) : int
+set_message(message:String):void
+get_message(void): String
+set_visible(visible : bool) : void
+get_visible(void) : bool
+set_color(color: Color):void
+get_color(void):Color
1
1
ButtonBar
size : int
isVisible : bool
buttons[] : Button[]
color:Color
+set_size(size:int):void
+get_size(void) : int
+set_visible(visible : bool) : void
+get_visible(void) : bool
+add_button(button : Button) : void
+get_button(id: int) : Button
+get_buttons(void) : Button[]
+set_color(color: Color):void
+get_color(void):Color
1
1
<<interface>>
WirelessCommunication
+send_data(data:TCPpackage): bool
+get_data(void) : TCPpackage
1
1
InformationBar
fontSize : int
information :String
zoomLevel : int
isVisible : bool
color: Color
+set_fontSize(size:int):void
+get_fontSize(void) : int
+set_information(info:String):void
+get_information(void): String
+set_visible(visible : bool) : void
+get_visible(void) : bool
+set_zoomLevel(zoom : int) : void
+get_zoomLevel(void) : int
+set_color(color: Color):void
+get_color(void):Color
1
n
Button
size : int
isVisible : bool
priority : int
color:Color
+set_size(size:int):void
+get_size(void) : int
+set_visible(visible : bool) : void
+get_visible(void) : bool
+set_priority(priority:int):void
+get_priority(void) : int
+set_color(color: Color):void
+get_color(void):Color

Figure 5.12

5.2.6.1 Processing narrative for component Screen

This component is responsible for keeping all data about the screen. This data is represented in MessageBox, ButtonBar, InformationBar classes and Map package. Another main duty of this component is refreshing the screen. Moreover, MessageBox implements WirelessCommunication interface to get message from the server.

5.2.6.2 Interface Description for component GUI Manager

The inputs of this component are coming from GUIManager that sets attributes of Screen package classes. Another input is message coming from server machine. The output of this package is modified and refreshed screen content.

5.2.6.3 Processing Detail for component Screen

Actually Screen package is a container package that keeps screen entities and their characteristics that can be seen at class diagram of this package. The main action Screen package performs is that refreshing the screen triggered by GUIManager. When GUIManager decides to change screen content, it calls refresh () method of the Screen class. Another process Screen Package performs is displaying message whenever a message comes.

5.2.7 GUIManager

Diagram for component GUIManager

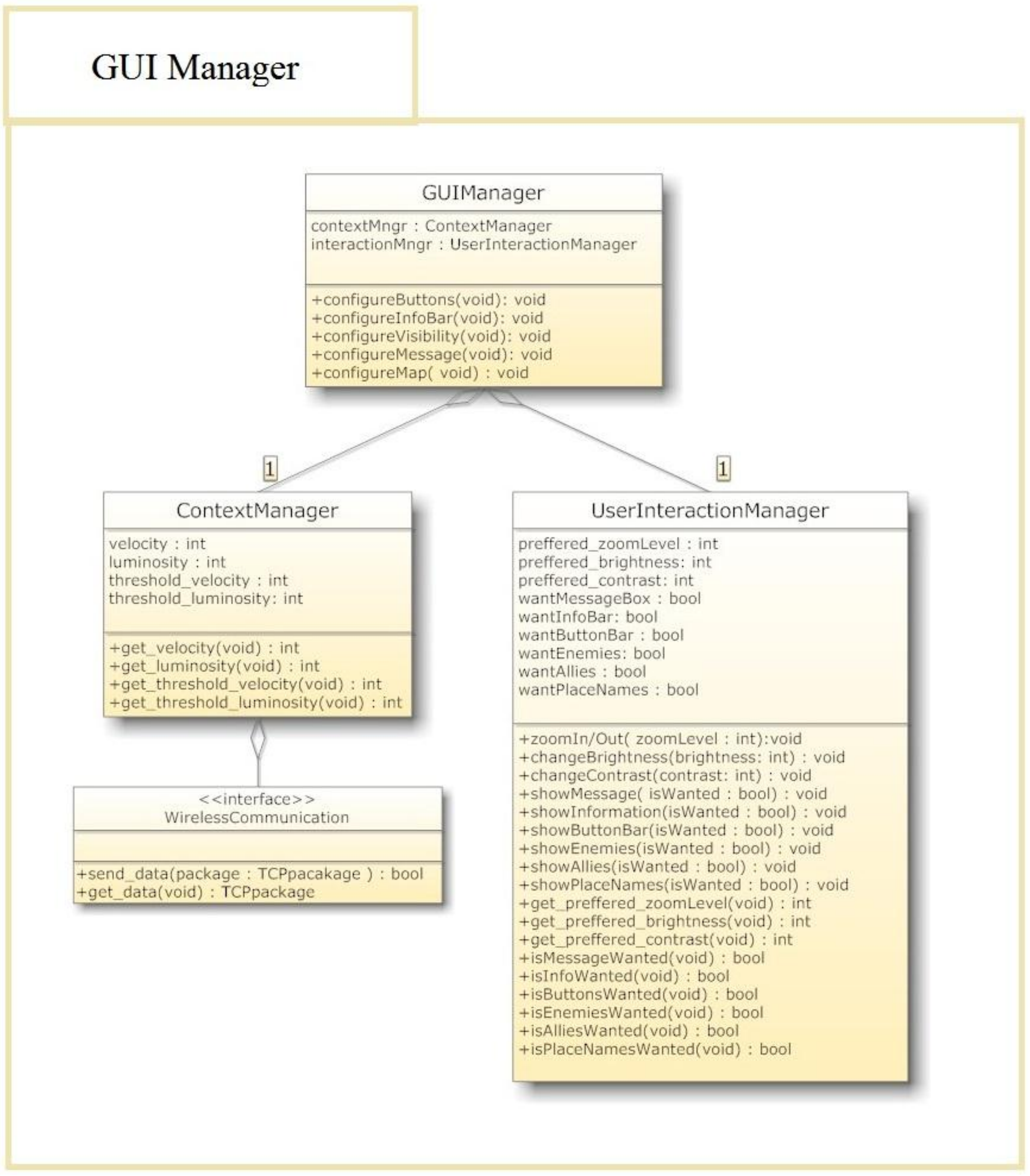

GUI Manager
GUIManager
contextMngr : ContextManager
interactionMngr : UserInteractionManager
+configureButtons(void): void
+configureInfoBar(void): void
+configureVisibility(void): void
+configureMessage(void): void
+configureMap(void) : void
1
1
ContextManager
velocity : int
luminosity : int
threshold_velocity : int
threshold_luminosity: int
+get_velocity(void) : int
+get_luminosity(void) : int
+get_threshold_velocity(void) : int
+get_threshold_luminosity(void) : int
<<interface>>
WirelessCommunication
+send_data(package : TCPpacakage) : bool
+get_data(void) : TCPpackage
UserInteractionManager
preffered_zoomLevel : int
preffered_brightness: int
preffered_contrast: int
wantMessageBox : bool
wantInfoBar: bool
wantButtonBar : bool
wantEnemies: bool
wantAllies : bool
wantPlaceNames : bool
+zoomIn/Out(zoomLevel : int):void
+changeBrightness(brightness: int) : void
+changeContrast(contrast: int) : void
+showMessage(isWanted : bool) : void
+showInformation(isWanted : bool) : void
+showButtonBar(isWanted : bool) : void
+showEnemies(isWanted : bool) : void
+showAllies(isWanted : bool) : void
+showPlaceNames(isWanted : bool) : void
+get_preffered_zoomLevel(void) : int
+get_preffered_brightness(void) : int
+get_preffered_contrast(void) : int
+isMessageWanted(void) : bool
+isInfoWanted(void) : bool
+isButtonsWanted(void) : bool
+isEnemiesWanted(void) : bool
+isAlliesWanted(void) : bool
+isPlaceNamesWanted(void) : bool

Figure 5.13

5.2.6.1 Processing narrative for component GUI Manager

This component is responsible for managing graphical user interface by considering user preferences, user’s motion and environment’s luminosity. UserInteractionManager class interacts with a user by GUI and keeps user’s preferences without applying them. Meanwhile, ContextManager class keeps getting information about the context from server by Wireless Communication interface. Then, GUIManager class combines all of the information and decides appearance of the screen.

5.2.6.2 Interface Description for component GUI Manager

ContextManager inputs are evaluated frame data that coming from the server machine. This input gives information about the context by stating velocity and luminosity values. UserInteractionManager inputs are coming from user via GUI. These inputs indicate user preferences about the screen content. User can decide additional zoom level, brightness and contrast, visibility of message box, information bar, button bar, enemies, allies, and place names on the map. The only output of this component is decided screen content that is going to be rendered.

5.2.6.3 Processing Detail for component GUI Manager

UserInteractionManager's duty is keeping user preferences to take into consideration later in GUIManager methods. When user presses a button, function that sets related attribute is called. For example, if user wants to show enemies in the map, clicks on the "Show Enemies" button. Then "wantEnemies" attribute is set as "true" by calling "showEnemies ()" function. Other preferences are kept in the same way. UserInteractionManager is activated with user interaction. On the other hand, ContextManager is always active to get velocity and luminosity values. If ContextManager gets valid velocity and luminosity values from the server, GUIManager starts deciding the new screen content considering user preferences and context change by calling configureButtons(), configureInfoBar(), configureVisibility(), configureMessage(), configureMap() functions.

- configureButtons () function arranges button numbers, button sequence in the button bar and change buttons' size and color. If user preference is hiding button, this function only hides button bar, doesn't achieve former operations.
- configureInfoBar () function arranges information bar with respect to font size, text color, amount of information displayed. If user prefers to hide information bar, this actions are not necessary to be performed.

- configureVisibility () function adjusts brightness and contrast of the screen. If user wants additional brightness and contrast, this preference is also taken into consideration.
- configureMessage () function arranges message box size , color and messages text size , color. In case of user preference is hiding the message box, these procedures are not performed.

- configureMap () function arranges map's zoom level, color range, amount of information on the map such as place names, allies and enemies. If user prefers to show or hide information on the map, these preferences are considered.

The algorithmic details of these functions will be provided in Detailed Design Report.

5.2.7.4 Dynamic Behavior for the component GUIManager

This sequence diagram shows just how the contrast changes. Other properties change in the same way and parallel to the contrast change.

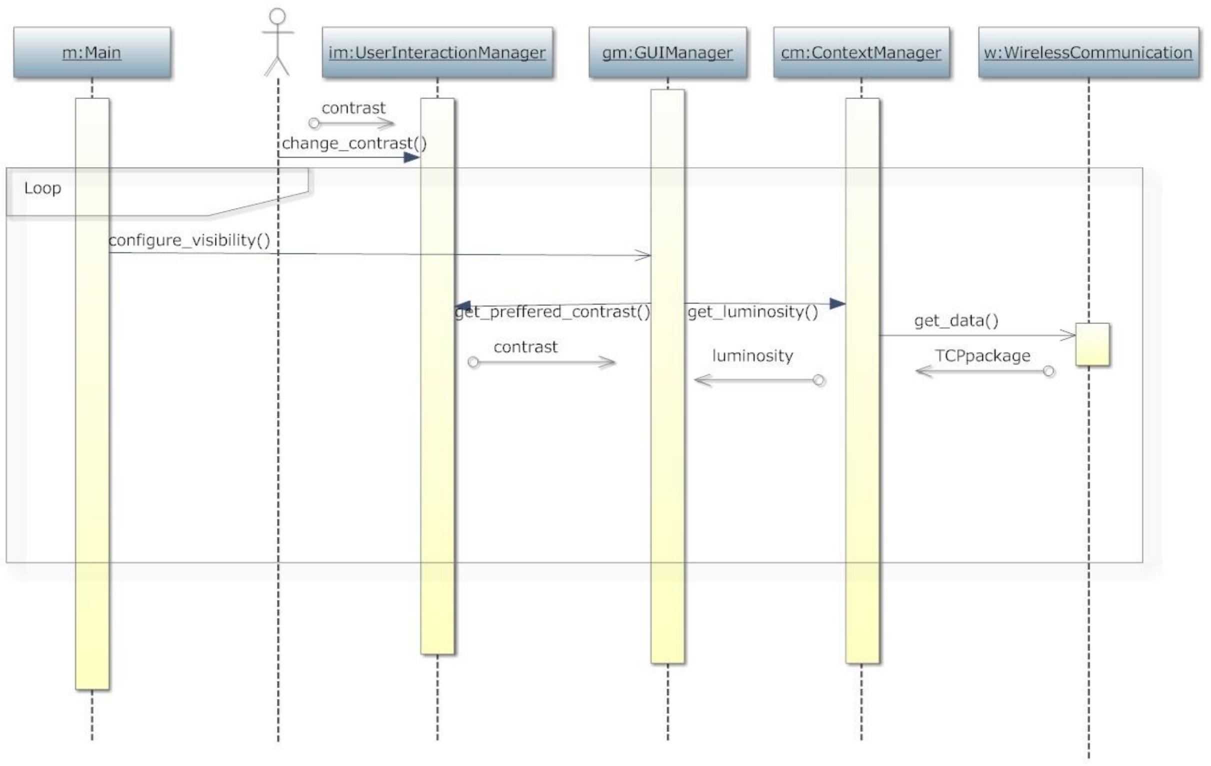


Figure 5.14

5.2.8 WirelessCommunication

WirelessCommunication is an interface class. It provides the connection between the server and mobile device.

Diagram for WirelessCommunication component:

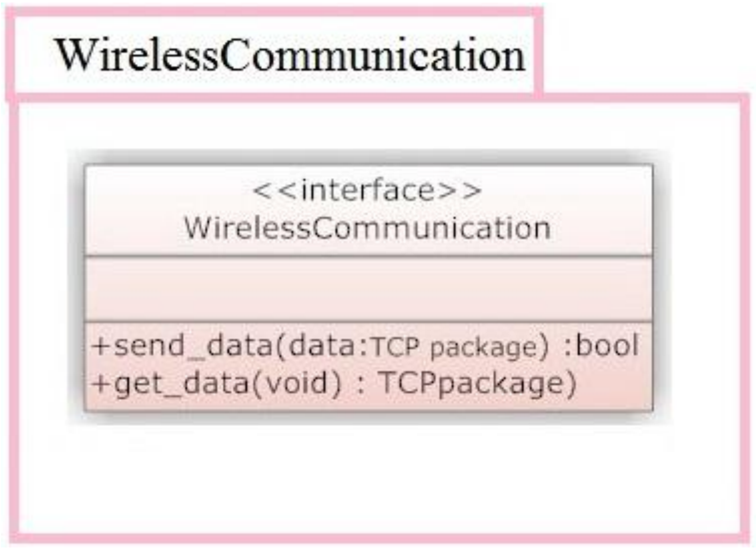

WirelessCommunication
<<interface>>
WirelessCommunication
+send_data(data:TCP package) :bool
+get_data(void) : TCPpackage)

Figure 5.15

6 User Interface Design

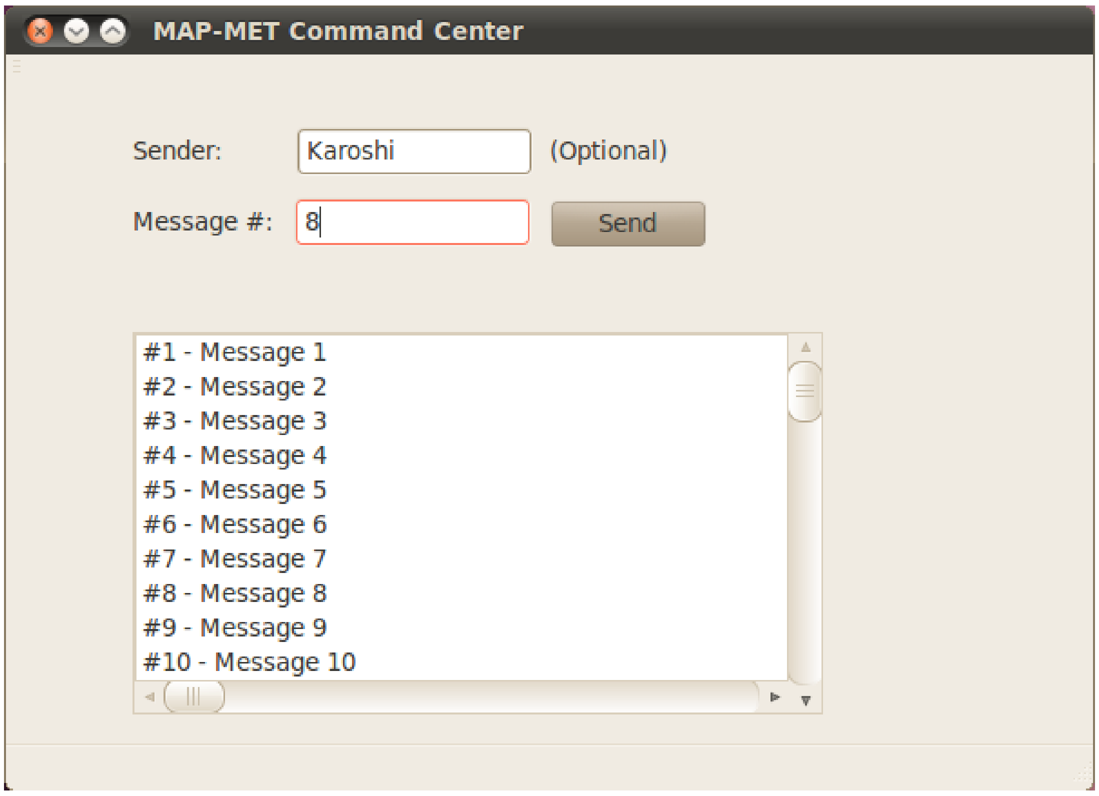
6.1 Overview of User Interface

The overall system can be mainly categorized under two user interfaces, one for sending message from the server side (Command Center) and another for the client side application. User interface of the server side has a simple structure. It lists all available messages and forward the message to client (mobile device) when user selects and sends one of the messages.

The complex part of the project is client-side user interface, since the main idea of the project is changing the user interface according to environmental parameters. Zoom level, colors, object sizes and object locations change depending on environmental factors. Therefore, too many user interfaces can be generated by the application. In Section 6.2, effects of the environmental changes on the user interface are shown with one example for each.

6.2 Screen Images

6.2.1 Sending Message from Command Center to Mobile Device

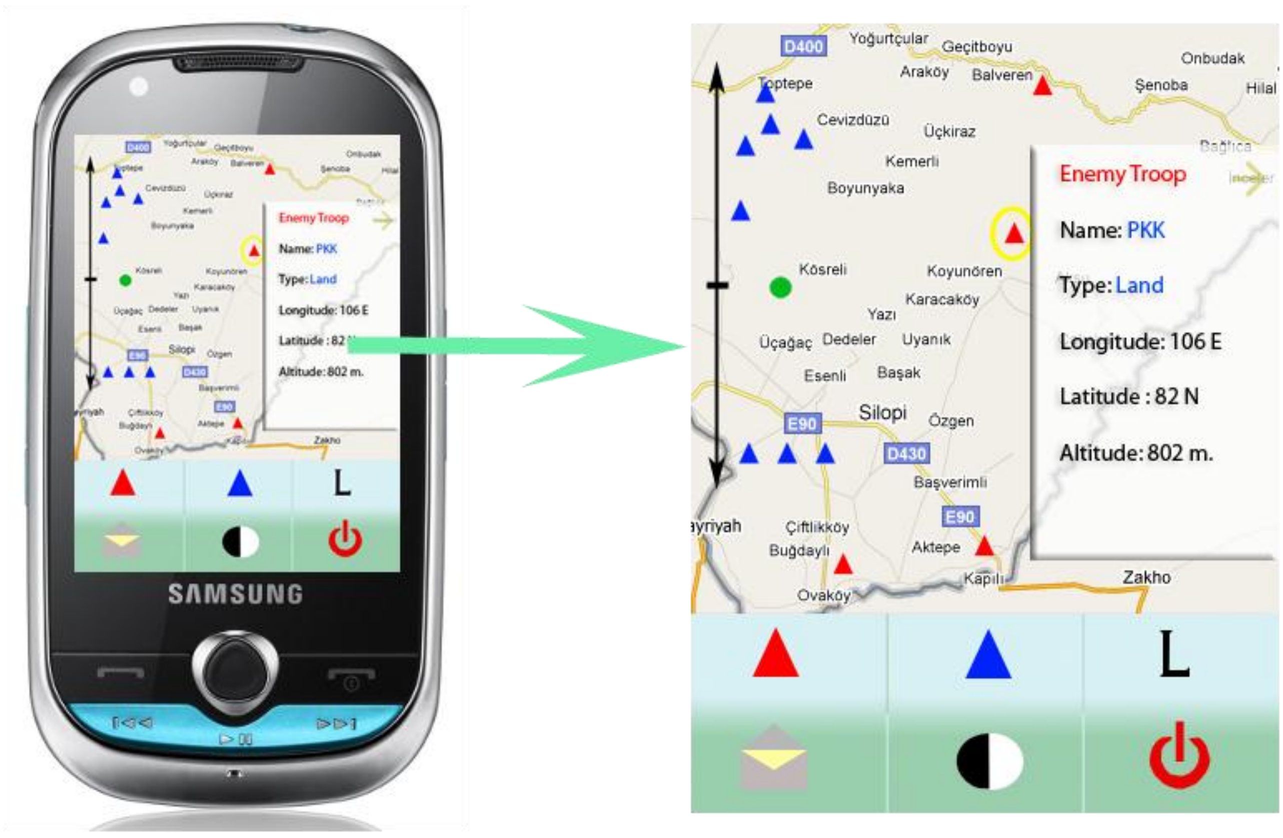


#1 - Message 1
#2 - Message 2
#3 - Message 3
#4 - Message 4
#5 - Message 5
#6 - Message 6
#7 - Message 7
#8 - Message 8
#9 - Message 9
#10 - Message 10

Figure 6.1

On this user interface all messages are shown to user. After message which is going to be sent has decided, user fills the box which is labeled as “Message #: “with corresponding message number. The user may want to specify the sender by filling the text box which is labeled as “Sender:” This operation is optional, if user does not fill sender information, the sender of the message will be set as “Command Center” by default.

6.2.2 Default User Interface for Mobile Device


Enemy Troop

Name: PKK

Type: Land

Longitude: 106 E

Latitude : 82 N

Altitude: 802 m.

Figure 6.2

This user interface is used for stable user when luminosity value is optimal. This user interface consists of four main parts, map, button box, zoom bar and information bar. On button box which presents at the bottom of the screen, there are six buttons available.

- *“Show/Hide Enemy”* button: The button which presents on the top-left corner of the button box, labeled with enemy sign.
- *“Show/Hide Ally”* button: The button which presents at the top-middle side of the button box, labeled with ally sign.
- *“Show/Hide Location” button*: The button which presents on the top-right corner of the button box, labeled with "L" sign.

- *"Inbox"* button: The button which presents on the bottom-left corner of the button box, labeled with a letter sign.
- *"Change Brightness/Contrast"* button: The button which presents at the bottom-middle side of the button box, labeled with contrast sign.
- *"Quit"* button: The button which presents on the bottom-right corner of the button box, labeled with quit sign.

Information bar is on the right-side of the screen. Details about selected point can be seen on it. There presents an arrow button on the top-right corner of the information bars to hide it.

Zoom-bar is on the left-side of the screen.

6.2.3 Default User Interface for Mobile Device with Hidden Components

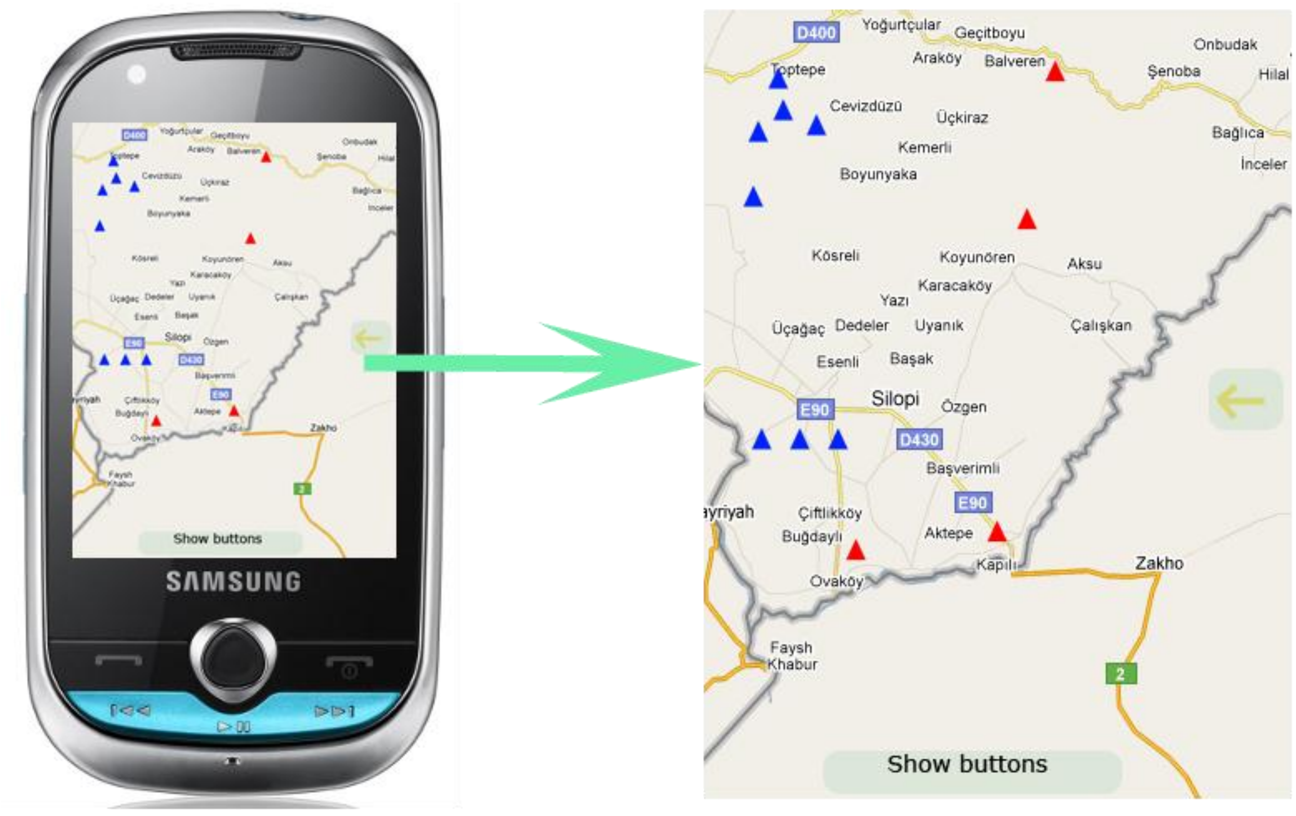


Figure 6.3

Zoom bar, button box and information bar can be hidden depending on context, user preferences and 30 seconds idle time. When these components are hidden two buttons reveal on the screen to recover them.

"Show Button Box Button" is at the bottom screen with a label of "Show buttons". Touching on this button recovers button box and zoom bar.

"Show Information Bar Button" is at the right-side of the screen with a label of left arrow. Touching on this button recovers information box.

6.2.4 User Interface for Mobile Device with Changing Colors

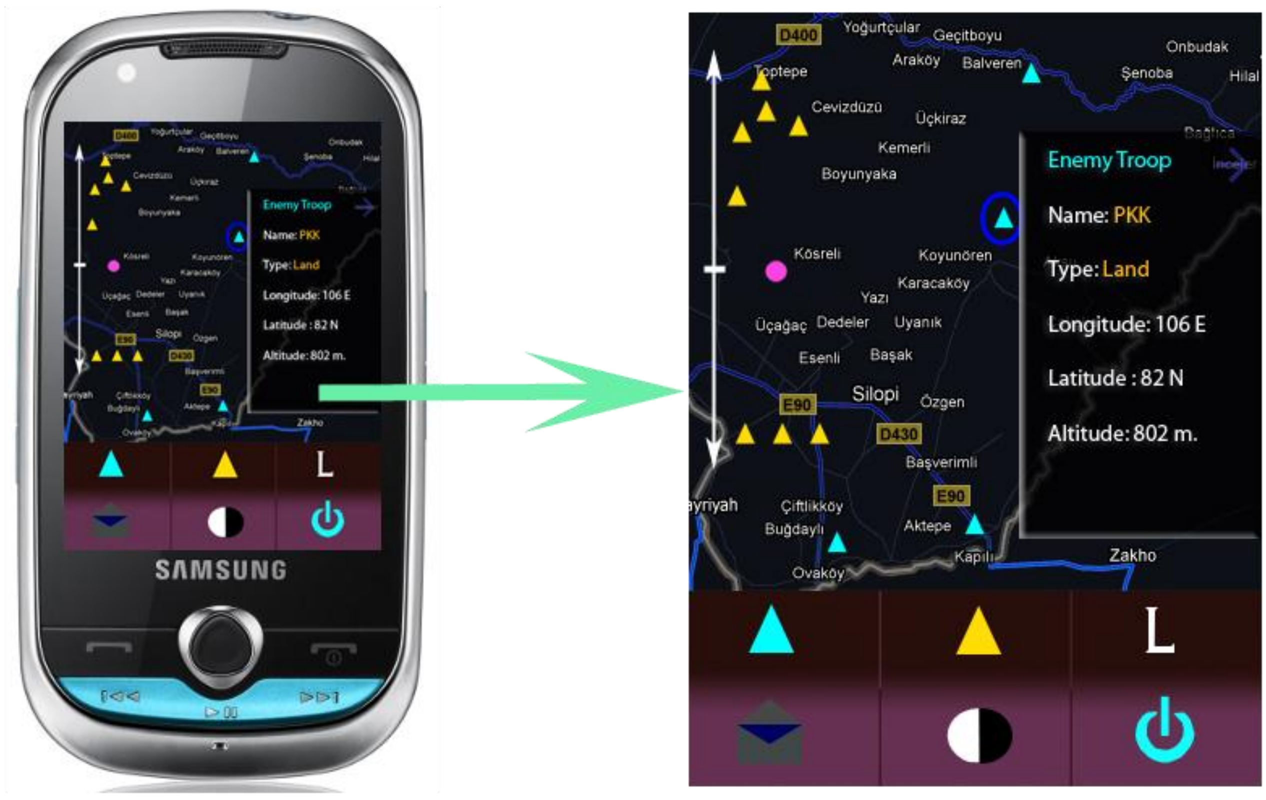


Figure 6.4

When luminosity value changes, colors on the user interface change accordingly. The user interface in Figure 6.4 is an example of how user interface reacts when the luminosity is low.

6.2.5 Brightness/Contrast Adjustment on Mobile Device

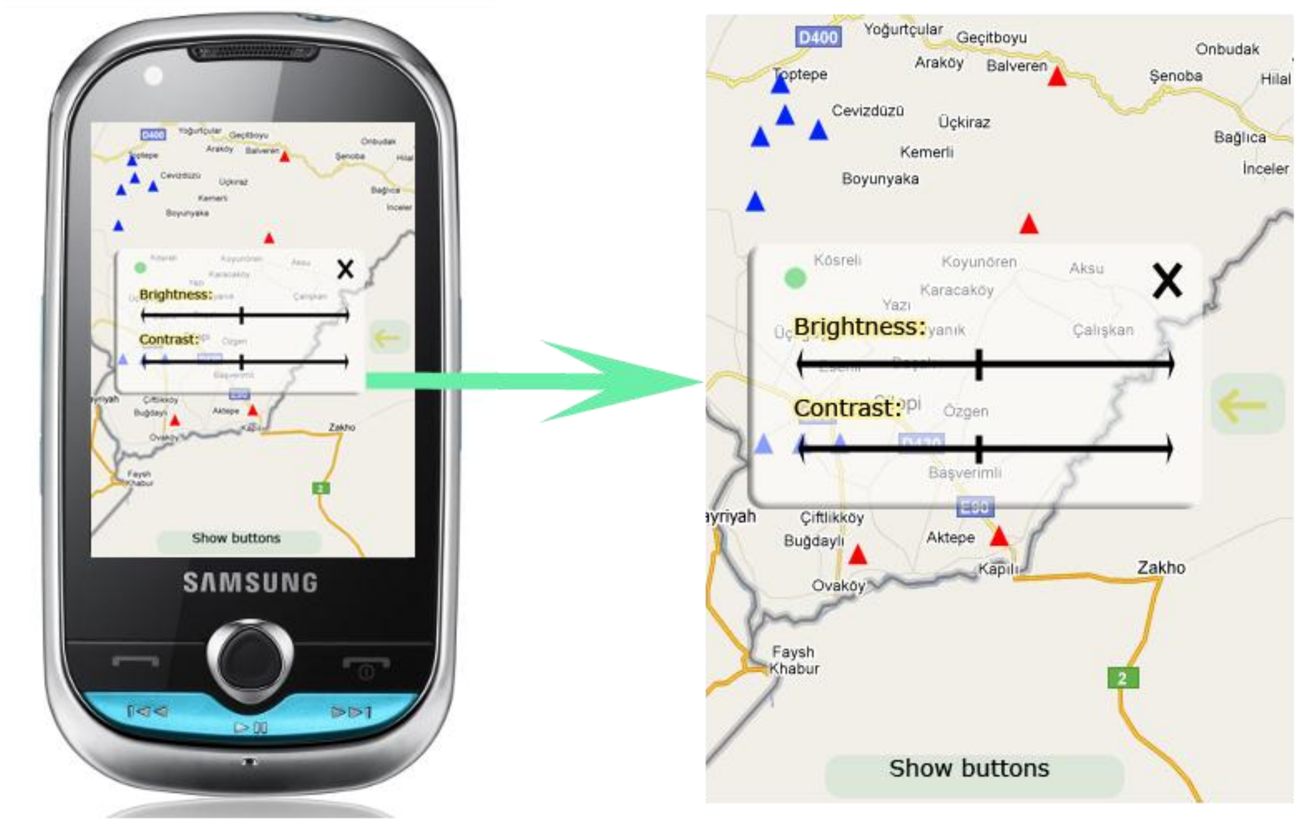


Figure 6.5

When *“Change Brightness/Contrast”* button is touched, a box appears on two scroll bars on it. Changing brightness and control values using scroll bars show immediate effect. When desired result is achieved, box can be closed using *"X"* button which is on top-right corner of the box.

6.2.6 Reading Message on Mobile Device

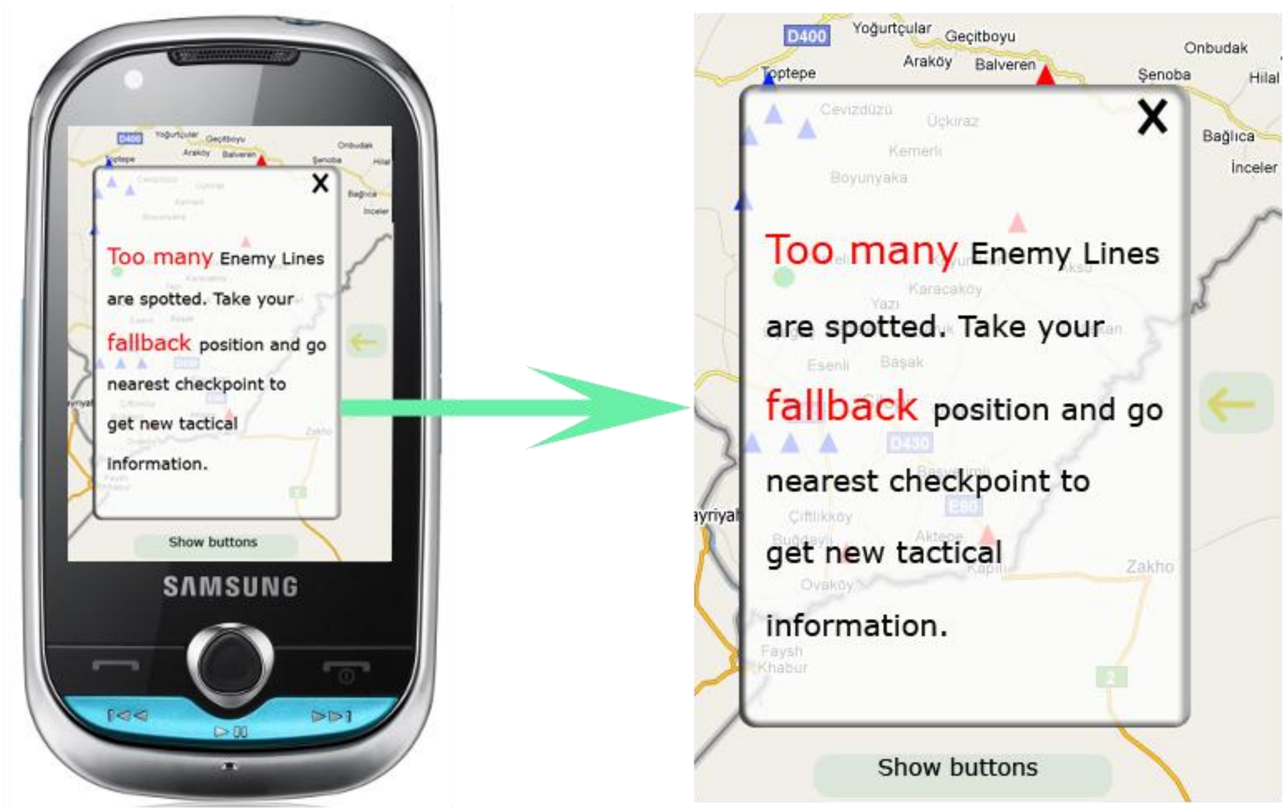

Too many Enemy Lines are spotted. Take your fallback position and go nearest checkpoint to get new tactical information.

Figure 6.6

When a new message is received or inbox button on the button box is touched, a message box appears showing the content of the message. The display of the message changes depending on the context. For example, while moving emphasized words becomes bigger and distinct.

6.3 Screen Objects and Actions

The input is gathered from the mobile device via touchscreen. On the screen, there exists three main areas, namely; map, button box and zoom bar. In addition to main components of the user interface, there are five auxiliary components: Message window, information bar, brightness/contrast window, show information bar button and show button box button.

Map: When a point on the map is touched, info bar reveals and the detailed information about that point is shown on the information bar.

Button Box: Button box arranges the buttons which serves different purposes.

- Using *"Show/Hide Enemy", "Show/Hide Ally"* and *"Show/Hide Location"* buttons, user is able to state his/her preferences. If the selection of the user is one of these to be hidden, then no information about that option is shown on the map. If user selects one of them to be shown, then that option will be shown on the map, but the detail level of this information will be decided by the application.
- Using *"Inbox"* button, user can view the latest message.
- *"Quit"* button is used to exit from application.
- Using *"Change Brightness/Contrast"* button, user can set his/her brightness and contrast preferences. When luminosity changes, application make adjustments depending on user preferences.

Button box is hidden by the application if user is idle for 30 seconds. When button box is hidden, show button box button reveals.

Zoom Bar: Zoom bar is used to get the preference of user about zoom level. Application changes this user specified zoom level according to the movement of the user and calculates the actual zooming level. Zoom bar is hidden by the application if user is idle for 30 seconds.

Message Window: This window opens when there is a new incoming message or *"Inbox"* button is touched. In this window, a message can be selected by touching on it for view it. To close the message box, user should touch on the "X" button which presents on the top-right corner of the message window.

Information Bar: This bar is not a main component of the user interface. When a point is selected on the map or *"Show Information Bar"* button is touched, this bar appears and shows the information about selected point if a new point is selected. If not, it shows the information about last used point. Information bar is hidden if user is idle for 30 seconds or it can also be hidden by touching the arrow button on the top-right corner of it. When information bar is hidden, *"Show Information Bar"* button is revealed.

Brightness/Contrast Window: When *"Change Brightness/Contrast"* button is touched, this window appears. On this window, there exist two bars. One of them is used for changing contrast, while other one is used for adjusting brightness.

Brightness/contrast window can be closed by touching “X” button which presents on the top-right corner of the brightness/contrast window.

Show Information Bar Button: When information bar is hidden, show information bar button is revealed. Touching this button reveals information bar.

Show Button Box Button: When button box is hidden, show button box button is revealed. Touching this button reveals button box.

7 Libraries and Tools

7.1 C++

C++ is a statically typed, freeform, multiparadigm, compiled, general purpose programming language. It is regarded as a middle level language, as it comprises a combination of both high-level and low-level language features. It was developed by Bjarne Stroustrup starting in 1979 at Bel Labs as an enhancement to the C programming language and originally named “C with Classes”. It was renamed to C++ in 1983. Some of its application domains include systems software, application software, device drivers, embedded software, high performance server and client applications. Therefore, C++ is selected for the server side implementation due to the performance constraints.

7.2 MATLAB

MATLAB (for **mat**rix **lab**oratory) is a numerical computing environment and fourth-generation programming language. Developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, and FORTRAN. We select MATLAB to achieve image processing computations easily. Another reason that affects our decision is that MATLAB provides easy binding with C++.

7.3 OPENCV

OpenCV is a library of programming functions mainly aimed at real time computer vision, developed by Intel and now supported by Willow Garage. It is free for use under the open source BSD license. The library is cross-platform. It focuses mainly on *real-time* image processing. We can use OPENCV functions because of the performance reasons and it can ease our job when extracting information about user’s speed and environment’s luminosity value.

7.4 Qt (framework)

Qt is a cross-platform application framework that is widely used for developing application software with graphical user interface (GUI) (in which case Qt is referred to as a *widget toolkit* when used as such), and also used for developing non-GUI programs such as command-line tools and consoles for servers. Qt is free and open source software. All editions support a wide range of compilers, including the GCC C++ compiler.We choose Qt to implement GUI of messaging application at server side.

7.5 Java ME

Java Platform, Micro Edition, or Java ME, is a Java platform designed for embedded systems (mobile devices are one kind of such systems) . Target devices range from industrial controls to mobile phones (especially feature phones) and set-top boxes. Java ME was formerly known as Java 2 Platform, Micro Edition (J2ME).

Java ME devices implement a profile. The most common of these are the Mobile Information Device Profile aimed at mobile devices. Mobile Information Device Profile includes a GUI, and a data storage API, and MIDP 2.0 includes a basic 2D gaming API. Since Java ME is portable, we choose this platform to implement mobile side application.

7.6 Java MicroEmulator

MicroEmulator is a pure Java implementation of Java ME in Java SE. MicroEmulator is licensed under LGPL so it is possible to link and distribute commercial software with its libraries. Example usages are stated below:

- Application demonstration in web browser applet
- Faster development of application in Eclipse
- Using standard java profiling tools to tune your application
- Creation of unit tests for J2ME application that runs during build process

We will use MicroEmulator to test mobile side application.

7.7 Transmission Control Protocol (TCP)

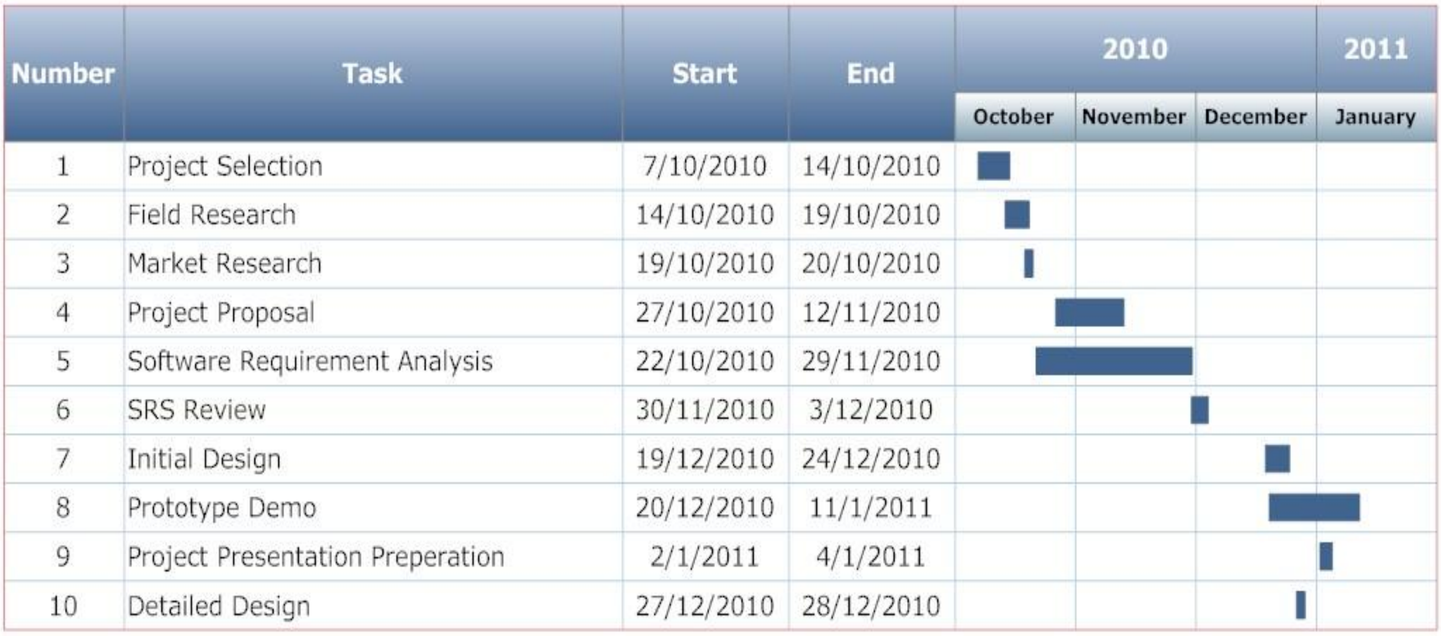
The Transmission Control Protocol (TCP) is one of the core protocols of the Internet Protocol Suite. TCP is one of the two original components of the suite, complementing the Internet Protocol (IP), and therefore the entire suite is commonly referred to as *TCP/IP*. TCP provides the service of exchanging data directly between two hosts on the same network, whereas IP handles addressing and routing message across one or more networks. In particular, TCP provides reliable, ordered delivery of a stream of bytes from a program on one computer to another program on another computer. TCP is the protocol that major Internet applications rely on, applications such as the World Wide Web, e-mail, and file transfer. We will use this protocol to achieve data transfer between server and mobile device.

7.8 Eclipse

Eclipse is a multi-language software development environment comprising and a plug-in system to extend it. It is written primarily in Java and can be used to develop applications in Java and, by means of the various plug-ins, in other languages as well, including C, C++, COBOL, Python, Perl, PHP, and others. The IDE is often called Eclipse ADT for Ada, Eclipse CDT for C and C++, Eclipse JDT for Java and Eclipse PDT for PHP. Eclipse CDT and Eclipse PDT will be used in the project development.

8 Time Planning

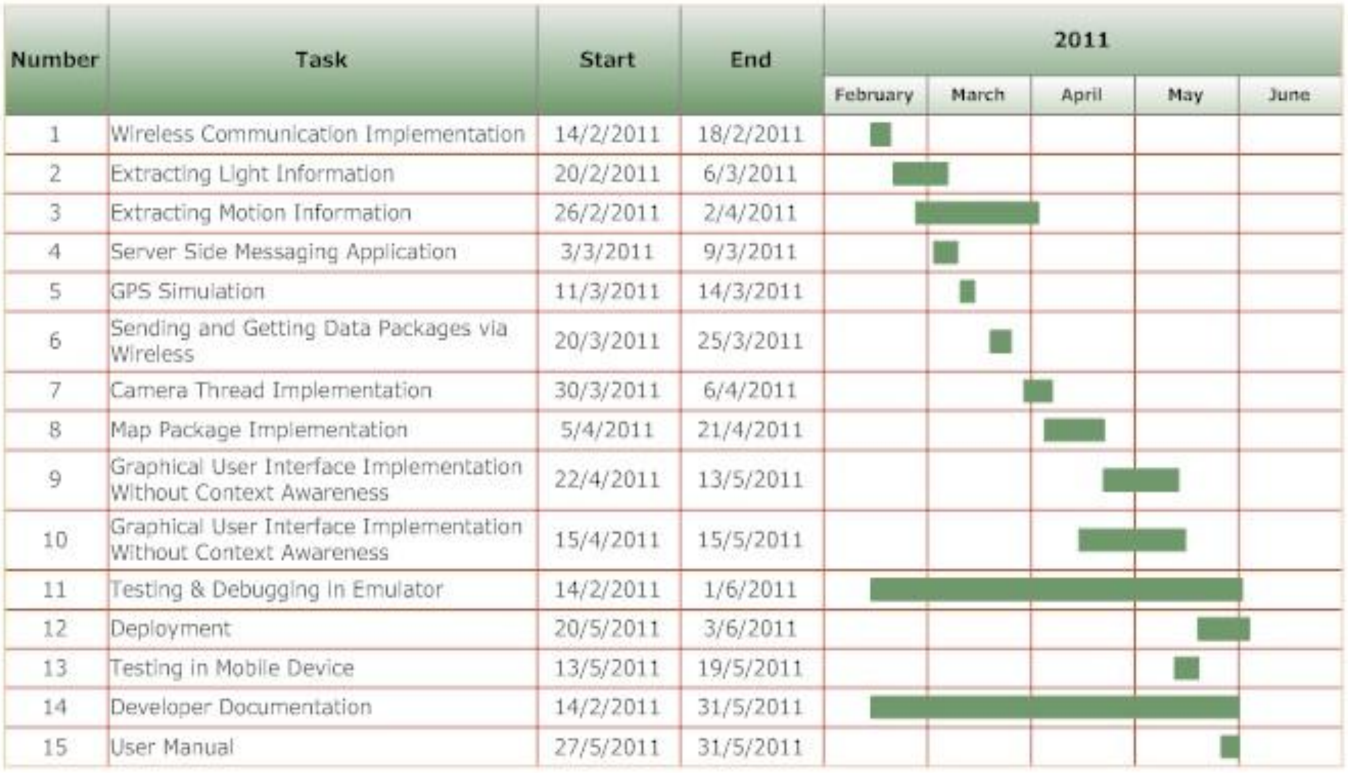
8.1 Term 1 Gantt Chart



Number	Task	Start	End	2010			2011
				October	November	December	January
1	Project Selection	7/10/2010	14/10/2010	■			
2	Field Research	14/10/2010	19/10/2010	■			
3	Market Research	19/10/2010	20/10/2010	■			
4	Project Proposal	27/10/2010	12/11/2010	■	■		
5	Software Requirement Analysis	22/10/2010	29/11/2010	■	■		
6	SRS Review	30/11/2010	3/12/2010		■	■	
7	Initial Design	19/12/2010	24/12/2010			■	
8	Prototype Demo	20/12/2010	11/1/2011			■	■
9	Project Presentation Preperation	2/1/2011	4/1/2011				■
10	Detailed Design	27/12/2010	28/12/2010			■	

Table 8.1

8.2 Term 2 Gantt Chart



Number	Task	Start	End	2011				
				February	March	April	May	June
1	Wireless Communication Implementation	14/2/2011	18/2/2011	🟩				
2	Extracting Light Information	20/2/2011	6/3/2011	🟩	🟩			
3	Extracting Motion Information	26/2/2011	2/4/2011	🟩	🟩	🟩		
4	Server Side Messaging Application	3/3/2011	9/3/2011		🟩			
5	GPS Simulation	11/3/2011	14/3/2011		🟩			
6	Sending and Getting Data Packages via Wireless	20/3/2011	25/3/2011		🟩			
7	Camera Thread Implementation	30/3/2011	6/4/2011		🟩	🟩		
8	Map Package Implementation	5/4/2011	21/4/2011			🟩		
9	Graphical User Interface Implementation Without Context Awareness	22/4/2011	13/5/2011			🟩	🟩	
10	Graphical User Interface Implementation Without Context Awareness	15/4/2011	15/5/2011			🟩	🟩	
11	Testing & Debugging in Emulator	14/2/2011	1/6/2011	🟩	🟩	🟩	🟩	🟩
12	Deployment	20/5/2011	3/6/2011				🟩	🟩
13	Testing in Mobile Device	13/5/2011	19/5/2011				🟩	
14	Developer Documentation	14/2/2011	31/5/2011	🟩	🟩	🟩	🟩	
15	User Manual	27/5/2011	31/5/2011				🟩	

Table 8.2

9 Conclusion

This Initial Design Report has been intended to explain how the system will be structured to satisfy the requirements. System components, interfaces and data required for the implementation phase have been briefly described. Gantt chart for the first and second semester has been also given. Detailed algorithmic description of components and data will be given in the Detailed Design Report.