SOFTWARE DESIGN DESCRIPTION

OF

COMRADE PROJECT

Prepared by;

SIGHT EFFECT

Burak GOYNUK	M. Gorkem GENC	Mustafa AKILLI	Tayfun ATES
1819374	1819366	1818897	1818970

Version 1.1

28.02.2015

Table of Contents

1. Overview

This document contains the system design information about Comrade System designed by Sight Effect Team. This document is prepared according to the "IEEE Standard for Information Technology – Systems Design – Software Design Descriptions – IEEE 1016 – 2009".

This Software Design Documentation provides a complete description of all the system design and views of the Project. The details of design issues, viewpoints, and concepts related to the Comrade system can be seen in this document.

1.1 Scope

The Comrade system is an application works on platforms which supports and runs Android operating system. In other words, the Comrade system is a mobile application, which is designed as a navigation system with several components in order to help people transport easily and safely; especially for blind people. The Comrade system will be used as interpreter system interacting with voice. In other words; it will take inputs in voice and return output with voice to it is used by blind people easily. Voice recognition is one of the essential parts of the Comrade system to complete interaction with blind people. Another important part of the Comrade system is the navigation part. It is designed for ease-of-use and efficient; since, the mobile platforms suffer lack of CPU capabilities and system resources. In addition to Voice Recognition and Navigation components, the Image Processing component of the Comrade system is also an essential part of the whole Comrade system. The main responsibility of the Image Processing subsystem is to transport blind people easily with using well-known object and line detection algorithms. The Comrade system also has some minor components to make user's transportation better. The taxi mode component, bus mode component, and message sending components or subsystems can be examples of such components. To sum up, the Comrade system is designed to be a navigation application runs on Android platform. It has many features and it has end-users which are both normal and blind people.

This document contains a structural overview of all modules, interfaces and data of this platform about the Comrade system. It also covers a detailed design of each module by giving information about the overall software architecture and the design methods for each module of the software product. A set of design views will be presented in order to support the design and development process. This document will serve as a guideline through the implementation phase. In other words, the developers who are responsible to develop Comrade System can easily use this document in order to have general understanding to system components, subsystems, and their interactions. The document also provides a well-understood of the main

structure of the project and how the implementation process will be done. The Comrade system's components, packages, system modules, interfaces and data used in the system can easily be obtained from this SDD document.

1.2 Purpose

The purpose of this document is to give a detailed description and visualization of the architecture for the Comrade system. These are namely a general description of the design elements, their interactions, the system's components and behavior before starting the code development phase. Another purpose of this document is to complete the needs of SRS. In addition, systems' deployment of the Comrade system, the packages, and its components, which complete the Comrade system, will be described in detail.

1.3 Intended Audience

The Development team of Comrade Project to implement purposes is intended audience of the system. Also, project managers, stakeholders, end users and testers are intended audience. Since the Comrade system has many end users to have transportation issues and wants to use Comrade Application; this document intended audience of this document is very large including both regular and blind people.

1.4 References

- IEEE. *IEEE STD 1016-2009 IEEE Standard for Information Technology – System Design – Software Design Descriptions.* IEEE Computer Society, 2009.
- *StarUML 5.0 User Guide.* (2005). Retrieved from http://staruml.sourceforge.net/docs/user-guide(en)/toc.html:

2. Definitions, Acronyms and Abbreviations

StarUML	An open source UML tool, licensed under a modified version of GNU GPL
UML	Unified Modeling Language
IEEE	Institute of Electrical and Electronics Engineers
GUI	Graphical user interface
SDD	Software Design Description
SRS	Software Requirements Specification
OOP	Object Oriented Programming

METU	Middle East Technical University
CEng	Computer Engineering
HAL	Hardware Abstraction Layer
I/O	Input Output
API	Application Programmer Interface
Dropdown List	GUI element which allows the user to choose one value from a list.
GUI	Graphical User Interface
IEEE Std 840-1998	Recommended practice for software requirement specification by the Institute of Electrical and Electronics Engineers
IEEE Std 1016-2009	Recommended practice for the content and the organization for a software design description by the Institute of Electrical Electronics Engineers
OpenCV	A real time computer vision library that is free for academic and commercial purposes
OS(Operating System)	It is software that manages computer hardware and software resources and provides common services for computer programs
APK	Android Application Package
dex	Dalvik Executables
GPS	Global Position System
CPU	Central Processing Unit

Table 1 – Definitions and Abbreviations

3. Conceptual Model for Software Design Descriptions

In this section, the conceptual model for this SDD will be described. The conceptual model consists of four main parts, these parts are basic terms and concepts of SDD, the context in which SDD is prepared and used, the stakeholders who use them, and how they are used. In other words, the conceptual model of SDD gives the description of concepts used in SDD and how this SDD is used. In these parts of SDD, the conceptual model will be discussed.

3.1 Software Design in Context

The Comrade project will be designed as object-oriented in a modular design fashion. Since it is mobile-based application and project, the maintainability and dependability is very important. With modularity and availability, new demands that stakeholders request can be provided easily and effectively. As the Comrade project will be implemented and designed in object-oriented approach, it will be automatically modular and adding new features to the system will be very easy. Because of being mobile application, Comrade Project is satisfying portability concept

easily. The Comrade project is run on platforms which has Android support. So, it can be said that the Comrade system provides portability with the help of Android Hardware Abstraction Layer. In other words, thanks to Android support, the Comrade project will be a platform-independent application. Performance is also another important issue because the Comrade system. Since the Comrade project will be used by many users, including blind people, it is essential to be real-time application. Especially for the image processing part, the Comrade system should be real-time to meet users' needs. Being a real-time application makes the Comrade project's performance measure critique. Thanks to used APIs and Android's open source development tools, this difficulty will also be eliminated. These constraints and problems faced through development of Comrade Project will be tried to solve with Comrade Design, which will be described later in this SDD.

3.2 Software Design Descriptions within Life Cycle

3.2.1 Influences on SDD Preparation

The main influence of this document is the SRS of Comrade project. Product perspective, functional, non-functional requirements and interface requirements that are the requirements in the SRS and also the demands of the stakeholders specify the design of the project.

3.2.2 Influences on Software Life Cycle Products

The agile method will be used for the system. The development team should meet regularly to communicate efficiently and apply agile methodologies. Since this is an Android application project, first thing is to do visualize the main pages or interfaces of the Comrade application. In order to do this, GUI components should be prepared for the stakeholders. Also prototype of the system should be ready for them. After getting feedbacks from stakeholders, the needs of the system must be filled. Moreover, in the first demo, the small partitions of each component of the application should be ready. To illustrate, the navigation component should be work efficiently, properly and correctly. It should have minimum the route for given two locations, or only the target is given as destination and source would be current location which is taken by Comrade System's navigation component automatically. In addition to navigation component, the voice recognition component behaves as input-output system for the Comrade system should be implemented and integrated to the Comrade prototype. After presentation this small prototype of the Comrade project, the feed backs will be taken and the flow and life cycle of the project will be changed if there is need for it.

3.2.3 Design Verification and Design Role in Validation

The specifications stated as in SRS should be designed correctly according to this SDD document. As a result, this document is a primary reference for the verification and validation of the system. This SDD also has an impact on the later stages such as test plans and testing stages. All system parts will be tested against these cases. It will be checked for whether the requirements fulfilled or not.

4. Design Description Information Content

4.1 Introduction

This document is designed to identify how the Comrade project will be designed and implemented. Also, in this section, it will be explained how the system architecture described.

4.2 SDD Identification

Comrade system's software design development parts are designed at the 20th of May, 2014. Comrade project's main purpose is to help solving navigation and transportation problem of both blind and normal people who face with. The Comrade System model and structure which is described in this document will be used for product and development iterations of the project. After the first term of the year, the prototype of the Comrade system will be demonstrated on January 20, 2015. Then, the development team of the Comrade system, who is Sight Effect Team, is responsible for updating the existing system according to feedbacks retrieved in demonstration; and they are responsible for completing the remaining parts of the Comrade system. The final product is planned to release on May 29, 2015. The final version of the Comrade system will be available on Google Play Store, and it can be downloaded without any payment. However, all rights of the end product are reserved. On the other hand, people will be encouraged to modify and improve features and functionalities of the Comrade system. Thanks to application store provided by Google, the Sight Effect team will keep track of issues related on the project. User comments and application usage data can be obtained from Google Play Store. Scope, references, context and summary can be found in section 1 in detail. Glossary and necessary terminology can be found in that section.

4.3 Design Stakeholders and Their Concerns

The developer team of the Comrade system is the Sight Effect team. This team also includes the tester since all tests related on the project will be done by Sight Effect. The stakeholders of the project include Sight Effect Team, METU CENG Staff and end users. Stakeholders' main concern on the Comrade project is about the real-time specifications on the project, its functionalities

and its verification. The voice recognition and responding part of the system is essential, since the Comrade system will be served to blind people. The navigation part is also important as the Comrade is an improved version of the navigation system. The details of these requirements and additional, further requirements are specified and can be found on the SRS document of the Comrade project.

4.4 Design Views

This Project will be implemented as a mobile application in object-oriented design with simple graphical user interface. Since there is also voice recognition part of the project, the end users can select any interface which they want and use it. Being a mobile application makes the Comrade project portable automatically and stakeholders can use it in very large scaled product set. The interface of the Comrade project is modifiable according to stakeholders' desire. Also, the stakeholders can add new features or remove the unwanted ones in project. In later parts of this document contextual, composition, interface, logical, interaction and state dynamics view will be explained. In the following sections each view will be detailed with descriptions and diagrams.

4.5 Design View Points

Software design descriptions explain some viewpoints that are context, interface, logical interaction and state dynamics viewpoints. Context viewpoint describes the relationship, dependencies and interactions between the system and its environments. It shows the expectations from the user in the system. The duty of users and the stakeholders are detailed. Moreover system boundary should be designated. Interaction between the interfaces is explained in interface viewpoint. This is also needed for the test cases. It includes the details of the external and internal interfaces. Also interfaces viewpoints clearly specify which inputs give what outputs. Logical viewpoint shows the structure of the classes and their relationships. Interaction viewpoints represent the sequence of events in the system. State dynamic views shows the state transitions with diagrams.

4.6 Design Elements

The design elements of the Comrade system will be introduced in this section. Their fields such as entities, attributes, relationships or constraints can be seen in this section.

4.6.1 Design Entities

4.6.1.1 Application Core System

This system includes the source code, the permissions of the application and several platform dependent issues. In this system, the calculations and computations are needed by Comrade Requirements will be done according to users' input and the result of these computations will be served to the user through I/O handling component.

4.6.1.2 Navigation System

This system can be considered as external library which is written by development team of Comrade Project. It will include basics of a navigation application such as taking current location or finding route for given two locations. Both functionalities of the navigation system will be called by application core system.

4.6.1.3 Image Processing System

The Image Processing system will also behave like navigation system. It provides the abstraction with camera interaction to the application. In the Image Processing system, basic line detection and object detection algorithms will be implemented and the functionality of this system will be used in application core system.

4.6.1.4 I/O Handling Component

The I/O Handling Component is responsible for providing user-application communication. Since there are two basic usage of the Comrade application by the end users; the I/O handling component should satisfy requirements in these two ways. Firstly, the Comrade application will be used by graphical user interface by normal people, in this case, the I/O handling component is responsible for managing buttons and touch screen of the mobile phone. Secondly, the Comrade application will be managed by voice control and it should return outputs to voice. In this case, the I/O handling component will use the API served by Voice Recognition/Responding Component.

4.6.1.5 Voice Recognition/Responding Component

The Voice Recognition/Responding Component is used by I/O handling component in order to keep user interaction with voice. Thanks to this component, the Comrade system will gain the voice-interpreter features. It takes users' voice as an input, parses it and converts to string. For voice responding, it converts given string by I/O handling to voice and return to user.

4.6.1.6 Android API Service

In Navigation System, Image Processing System, I/O Handling Component and Voice Recognition/Responding Component, the APIs and library provided by Google to Android developers will be used in order to make Comrade more efficient.

4.6.1.7 AIDE

AIDE is stands for Android IDE. It is designed for Android developers and it provides desired libraries for developing Android applications. All functionalities and desired libraries will be used in Comrade System.

4.6.1.8 Java Program

Many subsystems of the Comrade system will be implemented in Java; which is a programming language. Its libraries and defined methods will be used in the development process.

4.6.1.9 Client Systems

The Client systems of the Comrade system will be Android devices. The clients who want to use Comrade Application will download APK of Comrade system. The Android OS will automatically setup the Comrade application and the executables with .dex extension will be worked. With this, the Comrade application will run over the mobile phones with Android OS.

4.6.2 Design Attributes

The attributes of the design entities will be described later in this SDD document.

4.6.3 Design Relationships

As stated in previous parts, the main subsystems and components of the Comrade system are; Application Core System, Navigation System, Image Processing System, I/O Handling System, and Voice Recognition/Responding Component.

Application Core System is responsible for making calculations according to states and inputs of the system. In this system, navigation system's, image processing system's, and I/O handling system's functions can be called according to their API.

Navigation system is used by Application Core system in order to handle GPS, location and navigation issues about the Comrade project.

Image Processing System is responsible for implementing and containing image processing algorithms. These methods are also called by Application Core System.

I/O Handling system is responsible for providing user-application connection. It should include both voice and graphical interface.

Hence; all I/O Handling, Image Processing and Navigation systems are used in Application Core. Other less important relationships between design entries will be described in Logical Viewpoint part.

4.6.4 Design Constraints

The design entities are described above. All described entities should be implemented. The relationships between these entities should also be preserved during implementation phase.

Since the Comrade is not static, a dynamic and maintainable project, all implementation should be fully object-oriented, reusable and maintainable. The implementation of the classes should be modular system and well-structured in order to understand by other developers and debugged easily.

In addition, since Comrade will be a real-time responding application, implementation of each method should be efficient in order not to waste CPU cycle.

4.7 Design Overlays

The interface of the Comrade system can be divided into two main groups, one is voice recognition and responding, and the other is graphical user interface. These interfaces are responsible for taking inputs of the system and giving outputs to the user. In other words, the main purpose of the interface of the system is getting inputs from the users and displaying results. The received inputs of the Comrade system is calculated and processed in Comrade Application's core system.

4.8 Design Rationale

The Comrade system's software is big with including many classes; each class includes many methods and different attributes. This situation causes complicated software architecture in the Comrade system. In this SDD, these classes are organized and subsystems of the Comrade system is grouped and managed efficiently. Each component is defined clearly in order to be guide for development team of the Comrade project.

Design choices are also made according to security, portability and maintainability. OOP concepts are also used for these concerns. It can be updateable according to stakeholders and users requirements. In addition, the system should be implemented in modular way in order to satisfy OOP related issues. With modularity, every component has its sandbox and these components do not affect each other. Each components of the system must be commented out so that the code is understandable and improvable.

4.9 Design Languages

Unified Modeling Language (UML) is used as design languages in this SDD document. In order to create UML products, StarUML is used.

5. Design Viewpoints

5.1 Introduction

Design viewpoints determine the conventions for a system including the architectural models, languages and notations. They are used in the realization part of the design descriptions and constraints of the Comrade application.

In this section, the following viewpoints of the software will be explained.

- Context Viewpoints
- Composition Viewpoints
- Logical Viewpoints
- Dependency Viewpoints
- Interface Viewpoints
- Structure Viewpoints
- Interaction Viewpoints
- State Dynamics Viewpoints

During the explanation of these viewpoints, text materials will be supported by charts and diagrams in order to increase understandability.

5.2 Context Viewpoints

Comrade application context viewpoint shows the all functions in the design. The context is defined by the elements that interact with the application like users. Also, it describes the relationships, dependencies, and interactions between the system and its environment. The actors of the system, who are blind people and tourists mostly, will be shown using UML language.

5.2.1 Design Concerns

Users of the system will use the system either through voice controls or commands and user interface. Voice controls for the use of blind people and the interfaces are for the other people. Blind people can use navigation to reach where they want. They use navigation by 3 different modes namely, taxi, bus or walking. In bus mode, the application takes the user to the closest bus station. In taxi mode, the application calls the nearest taxi station and calls the taxi. Also the application checks if the taxi driver defrauds the user or not. The navigation part they can start or stop object detection to learn whether there is an object on the road or not. They also get their battery status, since battery status is very important because of using internet and camera. Moreover they may send SMS of their location to their friends or relatives. All of these are done by through voice commands. The other people use the application through interfaces. Although they may only use the navigation part, they can use all of the three methods that are bus, taxi and walking mode.

5.2.2 Design Elements

One of the major design entities is the group of the actors of the Comrade Application which are end users. The end users of the application can be anyone who downloads through the Google Play Store. The system is not an online user interactive system therefore the user gets the output of the system through the android core functionalities and google play services. However, the user actions are also important because their commands or clicks may trigger the other functionalities.

Another group of design entities is stakeholders of the application. The Comrade Application is a mobile application project conducted in 4 METU CENG Students. The stakeholders are these four people.

5.2.3 Example languages

In the figure 1 the context or block diagram can be seen and in figure 2 the actors of the system and the functionality of the system is illustrated. All the use case descriptions were detailed in the SRS documentation of this application.

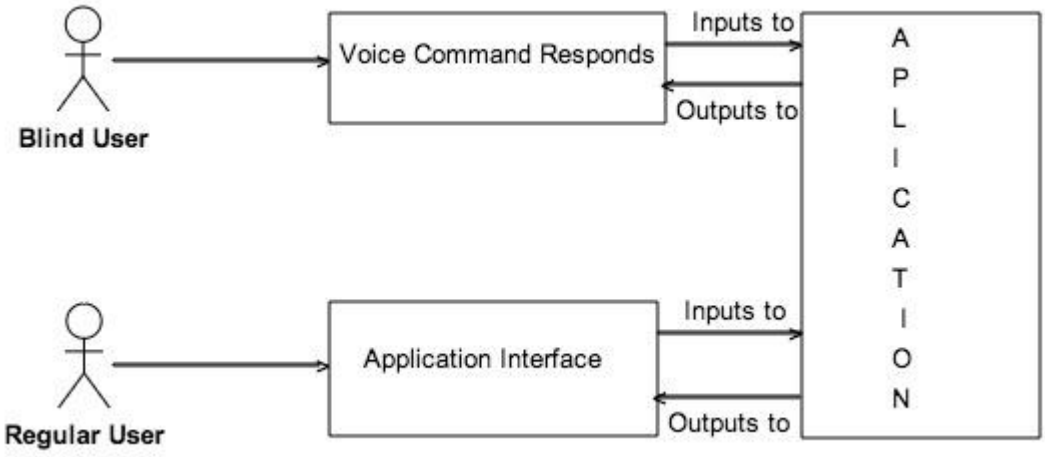

Figure 1 – System Context Diagram

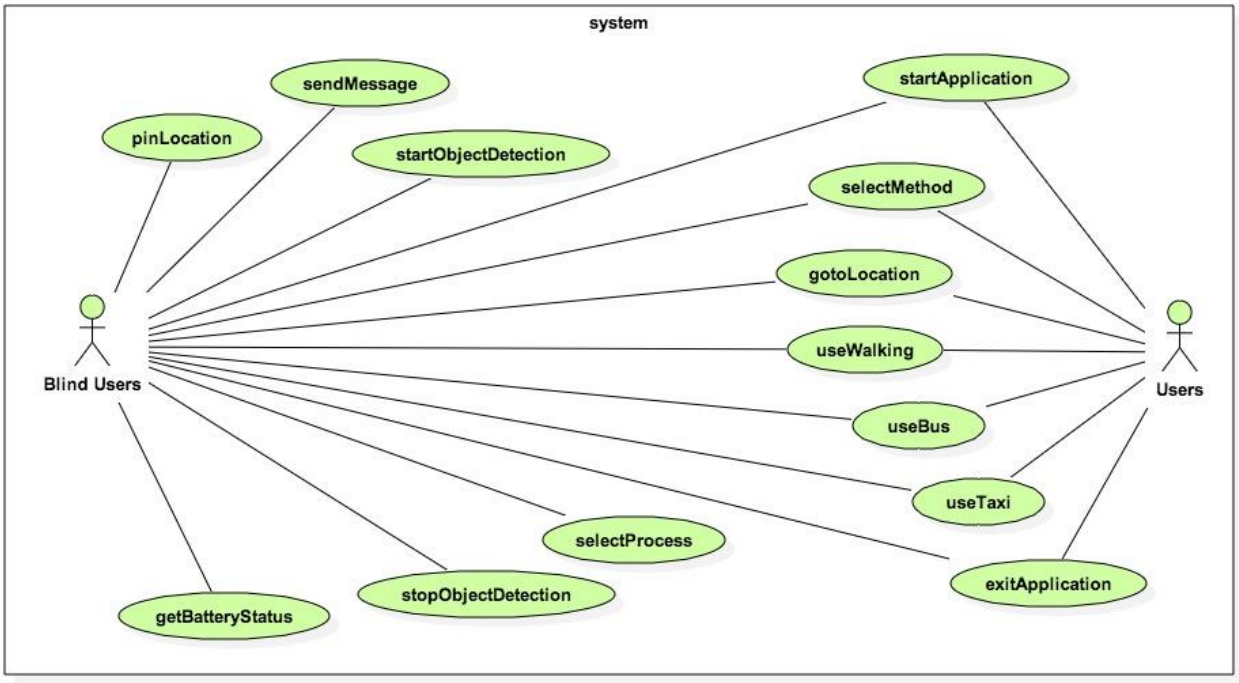

Figure 2 – Use Case Diagram

5.3 Composition Viewpoints

This section explains the information about Comrade Application components and their connections or relationships with each other.

5.3.1 Design Concerns

In this viewpoint, necessary information is provided programmers for controlling and planning the system. System components and their interconnections are described in UML Component Diagram. The system will be integrated to larger system. This kind of subsystem level illustration can be used for assembling components, cost estimation, not necessary for this application and schedules in terms of development effort.

In the figure 3 the composition diagram can be seen, system components such as libraries, packages, files and their interconnections are illustrated. Comrade application integrates Google Play Services and OpenCV. Google Maps API v2 is used for getting location of the user and finding path between 2 locations. Also Google Voice API is used for both voice to string and string to voice conversion. Moreover OpenCV libraries and functions are for obstacle detection.

5.3.2 Design Elements

The all necessary components and their interconnections are detailed in the subsections of this section.

5.3.2.1 Google Play Services - Function attribute

Google Play Services provides many APIs. However we need two of them namely, Google Maps API and Google Voice API. Maps API provides us to get location of the user and to get path between the location of the user and the target place. Latter one provides us to convert sting to voice and vice-versa.

5.3.2.2 Google Play Services - Subordinates attribute

The two APIs, mentioned in the previous section, composed of many functions and libraries that are referenced for our project. For each need there are some functions in these APIs.

5.3.2.3 OpenCV - Function attribute

OpenCV provide us to detect obstacles if they are on the yellow lines.

5.3.2.4 OpenCV - Subordinates attribute

OpenCV is also a big library that is composed of many functions. These functions take the pictures and divide into frames that we can use or manipulate.

5.3.2.5 User Interface - Function attribute

User Interface is for other than blind people. They can use basic functionalities of the Comrade application via these user interfaces.

5.3.2.6 User Interface - Subordinates attribute

-

5.3.2.7 Database - Function attribute

There is no database for this application; however this application stores some data on the external storage of the mobile phone.

5.3.2.8 Database - Subordinates attribute

-

5.3.3 Example languages

In the figure 3 the component diagram is seen. The components of the application and how they interact to each other is illustrated. Figure 4 shows the hardware components used to deploy software components for this application.

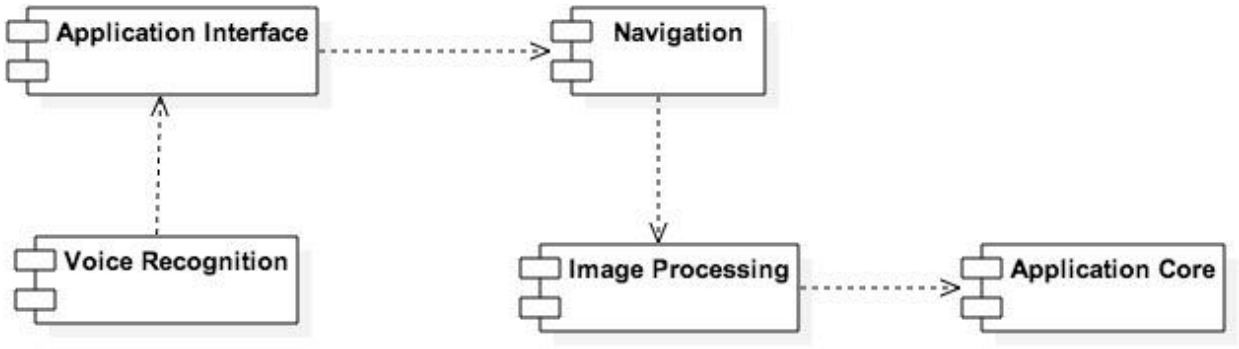

Figure 3 – Component Diagram

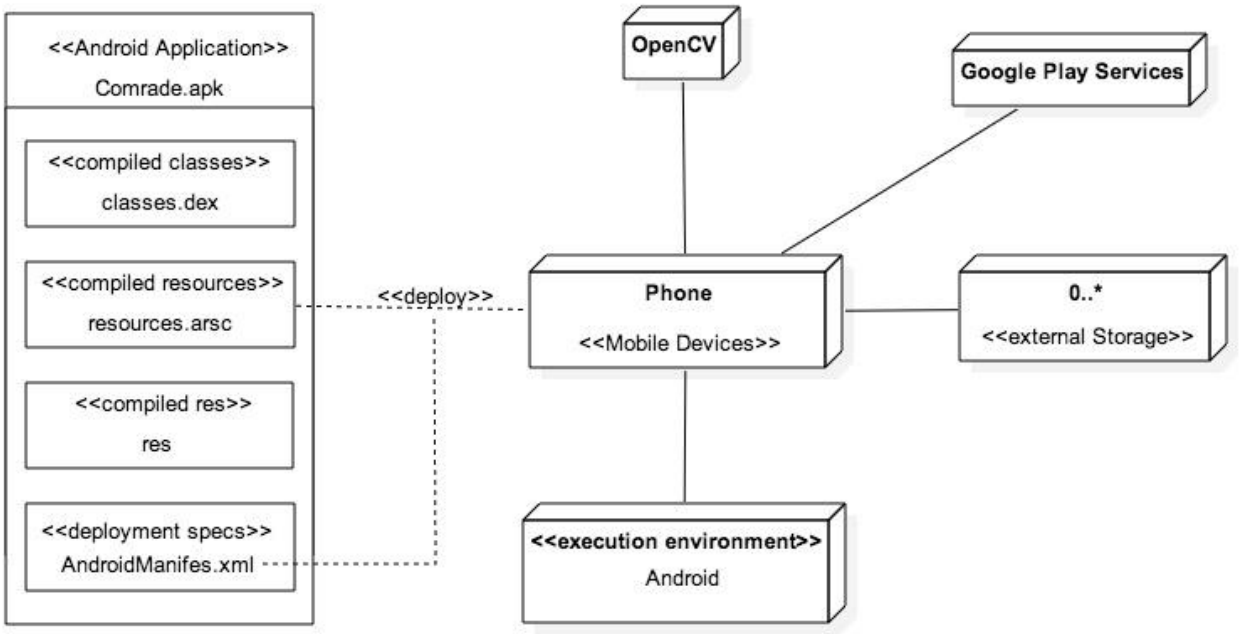

Figure 4 – Deployment Diagram

5.4 Logical Viewpoints

This viewpoint explains the static structure of system with entities, interaction among entities, namely classes and which are used in Comrade System. The relationships between these classes are explained in detail in this part of SDD. There are 11 classes in the system; the details of the attributes used in these classes can be seen in this viewpoint.

5.4.1 Design Concerns

Organization of the system classes will be shown in this part. All classes are designed according to SRS and the previous part of the document. These classes can be changed in the code development part. The relationships between classes and the attributes or entities of classes are visualized through UML in figure 5.

5.4.2 Design Elements

In this part the details of classes is going to be explained.

Property Class:

Property Class will keep all device specific data. This class will take information from device through Android OS and will keep related specifications on related attributes.

The attributes are listed below:

- *phoneSpec:* This is a fixed size list containing the string for device specification. The camera specification, device memory size, device CPU, and Android version of the device will be kept in the list in this order.
- *batteryStatus*: This will be an integer to keep battery status of the device. For example, this variable will be 75 for 75% charged situation.

The only method of this class except its constructor is getBatteryStatus method.

- *getBatteryStatus*: This function sets the batteryStatus variable after getting the value from Android API.

VoiceRecognition Class:

This class is used for transforming user input, voice to desired format for parsing, which is text. Also, the voice response of the application will be constructed here with converting text to speech.

The attributes of this class are listed below:

- *command*: This is a string typed attribute containing the converted string from user voice input.
- *response*: This variable is the statically or dynamically filled string for responding user with voice. In other words, this is the string to be converted voice for giving user to voice response.
- *responseVoice and commandVoice*: Voice is a voice(or speech) typed attribute of the class. Note that voice (or speech) type is defined on Android SDK. These variables will be used for both input voice and voice response according to application's state.

The methods of this class are listed below:

- *setVoiceCommand*: This method takes voice input and sets the voiceCommand variable. This will converted to string by speechToText method.
- *setResponse*: This method takes the string input that is the response of our application, and sets the response string variable.
- *TextToSpeech*: TextToSpeech method makes string to voice conversion. It converts response string to responceVoice variable.
- *SpeechToText*: SpeechToText method makes voice to speech conversion. It converts commandVoice variable to command string.

ImageProcessing Class:

This class will be used as image processing component for the application.

The attributes of the class can be listed as:

- *isStarted*: This is a Boolean variable to keep whether image processing progress is started or not.
- *image*: This is a list of OpenCV images to be processed.

The methods of this class are listed below:

- *StartDetection*: This method starts the object detection and sets the isStarted variable to true. This method opens the camera and fills the image list.
- *StopDetection*: This method stops the object detection and sets the isStarted variable to false. This method closes the camera and clears the image list.
- *CheckYellowLine*: This function checks whether the taken photos include yellow line or not. If the yellow line does not found, it returns false. On the other hand, it calibrates mobile phone with informing the user.
- *CheckObjectOnYellowLine*: This function checks whether there is an object on yellow line or not by using the OpenCV functionalities.

User Class:

The User Class is the abstract parent class of the all RegularUser and BlindUser classes.

The attributes of the User Class are:

- *userType*: This is a string to hold current user type of the application, which can be “Blind User” or “Regular User”.
- *method*: This variable is a TransportationMethod type which holds the all information about navigating user.
- *property*: The property variable is the instance of Property Class holds specifications of device, where application is run currently.
- *voice*: A VoiceRecognition object to be used interaction application with user.
- *image*: An ImageProcessing object to be used on application’s image processing part.

This class has no additional method.

BlindUser Class:

This class is special for the blind users and it inherits the user class in order to use its functionalities.

This class has only one attribute that is:

- *pinnedLocation*: This variable is the list of strings and keeps the pinned locations by the user.

The BlindUser Class has two methods they are as follows:

- *pinLocation*: The pinLocation method takes the user location as input, in string format, and appends this location to pinnedLocations list.
- *sendSMS*: This method takes two inputs, which are, the telephone number as a string and the user location as string. After, the sendSMS method sends SMS the taken location to input telephone number.

RegularUser Class:

This class has no any additional fields; it is used to determine type of the user. This class inherits from user class and it uses the fields of parent class. Thanks to this class, the user type variable can be set.

Transportation Class:

Transportation class is the main class for keeps all data related transport user with selected method.

The attributes of this class are the navigation and transportationType.

- *navigation*: This is one instance of the Navigation class which keeps the essential information for transporting user.
- *transportationType*: This is a string typed variable which keeps the type of transportation. Its value can be “taxi”, “walking” or “bus”.

The methods of this class are listed below:

- *findNearest*: This method returns the nearest location of the taxi or bus station according to transportation type.
- *hasArrived*: This method checks whether the user arrives the target location or not by comparing his/her current location with target location.

The Transportation class has three subclasses used for selected method. These are Taxi, Bus and Walking classes.

Taxi Class:

This class is designed to represent user transportation by taxi. The necessary fields and methods for this type of transportation are defined in this class.

The attributes of this class are listed below:

- *taxiStationID*: It is an integer to keep the nearest taxi station id. Note that, the information of taxi stations is embedded in application and any specific taxi station and related information can be retrieved by its id.
- *taxiNumber*: This variable keeps the phone number of the nearest taxi station as a string.

The methods of this class are as follows:

- *call*: There is no input for call method. This method calls the number attribute which contains the phone number of taxi station whose id is kept in taxiStationID variable.
- *checkDefrauded*: This method checks whether the user is defrauded or not by checking the successor locations of the taxi. In case of a defrauded occurs, the application will inform the user.

Bus Class:

This class is used when user selects the bus type of transportation. The necessary fields and methods for bus transportation are defined in this class.

This class has only one attribute that is:

- *busStationID*: This is and integer that keeps nearest bus station's id. Note that, the information of bus stations is embedded in application and any specific bus station and related information can be retrieved by its id.

This class has method named:

- *getStationLocation*. This method looks for bus station's address whose id is saved in busStationID and returns its address in LatLng (Latitude-Longitude) form.

Walking Class:

This subclass has no any additional field. This class inherits functionalities and fields from transportationMethod class. This class is necessary for the user who selects walking type of transportation.

Navigation Class:

The Navigation Class has composition relationship with Transportation class.

The Navigation Class' attributes can be listed as:

- *distanceToTarget*: This is a float typed value containing distance from source to target location in km unit.
- *source*: This is the field containing source location in string form.
- *target*: This is the field containing target location in LatLng form.
- *currentLocation*: This is the field containing current location in string form.
- *path*: The type of this attribute is PolygonList, and this type is served in Google Maps API. The path attribute keeps the polygons to show path on a map, on mobile phone's screen.

The methods of the Navigation Class are listed below:

- *LatLngToString*: This method converts the computed latitude longitude typed variable to human readable string, which containing address.
- *StringToLatLng*: The StringToLatLng method converts strings which keep the address information to LatLng data type in order to use Google Maps API.
- *getLocation*: This method finds the current location and saves it in string form in the currentLocation variable. The Google Maps API returns location in LatLng form. In order to save this as string, it uses the LatLngToString method defined above.
- *setTarget*: This function takes string address as an input and convert to LatLng form with using StringToLatLng method and save the calculated value to the target variable.
- *getPath*: The getPath method finds the polygons from source address to target address, and, saves these polygons in the path variable in order to generate path from source to target.
- setTransportationType: This method takes a string argument for transportation type and sets the variable transportationType which is in the Transportation class.

5.4.3 Example Language

Comrade system's class entities, which are described above, can be observed from below, in the class diagram of the related system. In addition, the object instances of the Comrade system, object diagram of the Comrade system can also observed in below. For representing these, UML diagrams are used.

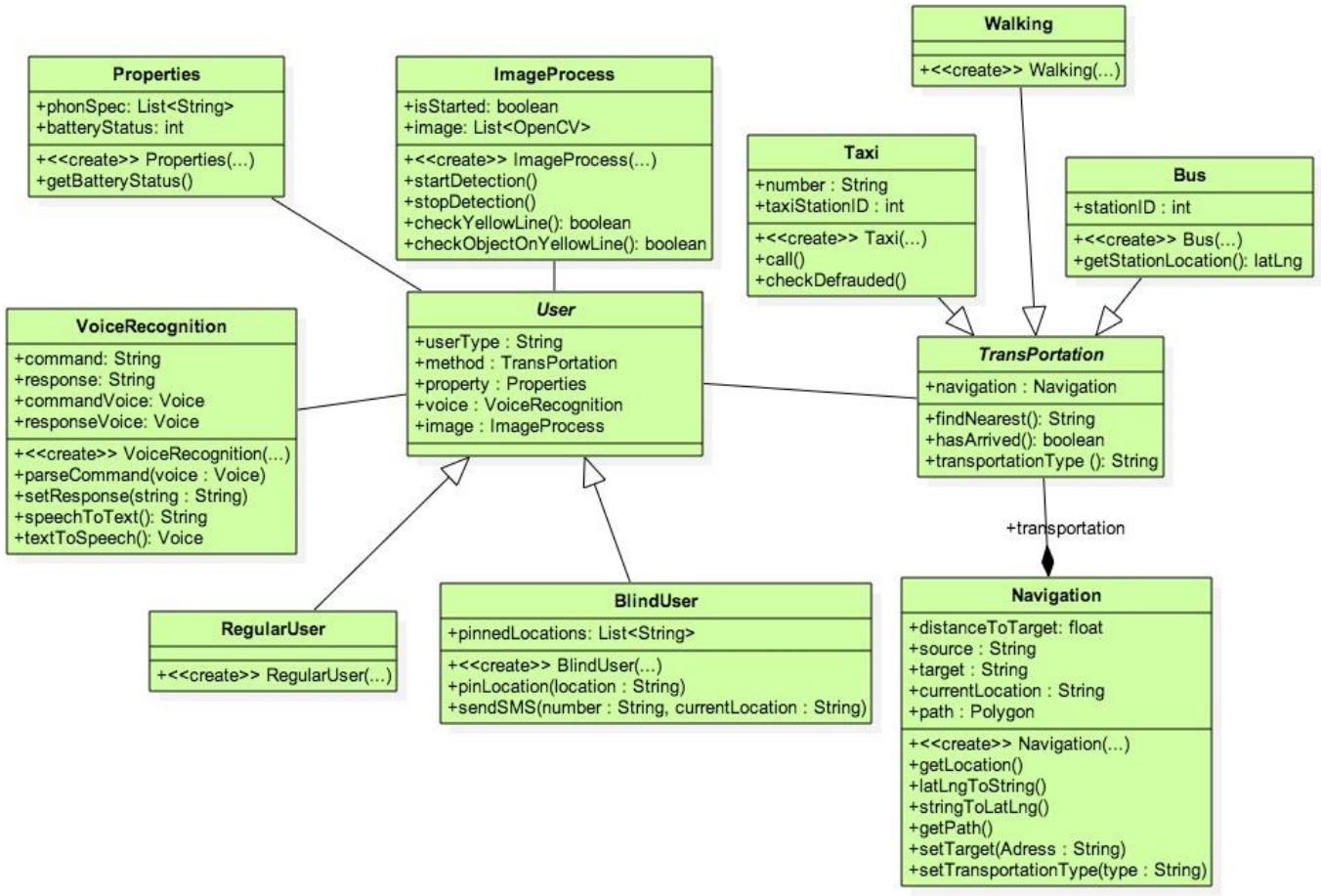

Figure 5 – Class Diagram

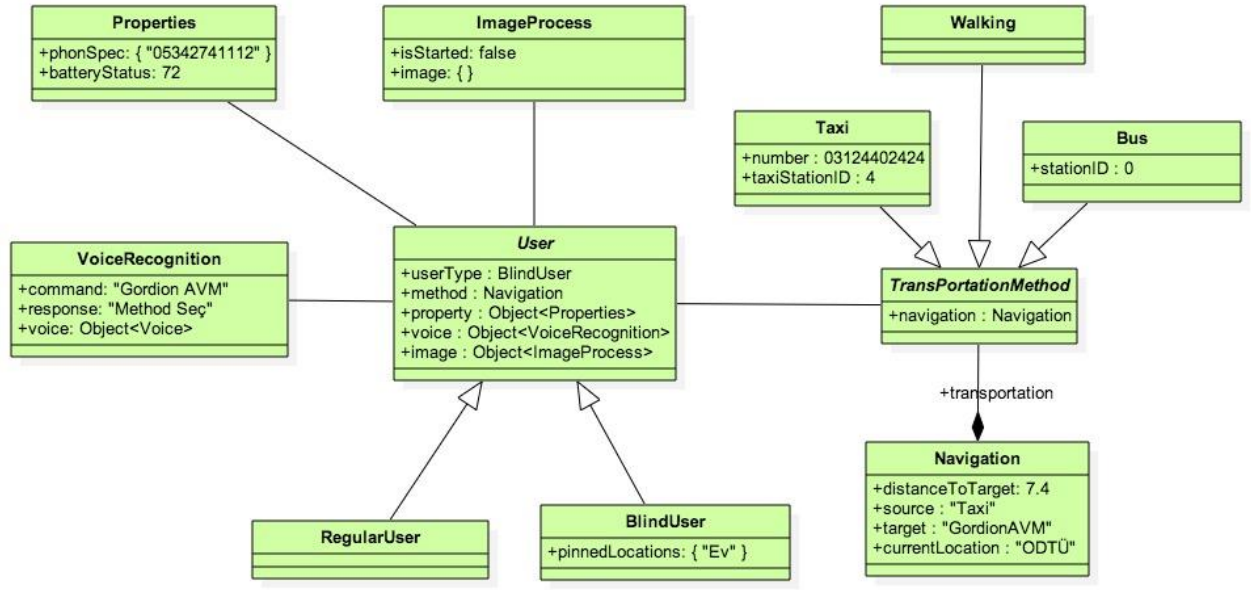

Figure 6 – Object Diagram

5.5 Dependency Viewpoints

This section provides the relationships of the components and how they interconnect to each other. Also, how the components are dependent to each other will be detailed as well as their requirements.

5.5.1 Design Concerns

This application has 4 main components. That are navigation, uses Google Maps API, voice recognition uses Google Voice API, and image processing part uses OpenCV. The fourth is the interfaces, designed for non-blind people. At the development phase each of the components can be developed without waiting for other component to finish. Before integration phase, each of the components can be researched, developed, tested by itself. After integration, the image processing component is depended on both voice and navigation component, and navigation component is depended on voice recognition component. This is because, the application waits for the voice commands and it's parsing, so that navigation part starts. In the navigation part the user can start obstacle detection through voice commands. Although all the other components depend on voice recognition component, voice recognition component is also depended on both navigation and image processing component, since it convert the string output of the navigation and image processing component. The components needs human interaction that are navigation and interfaces components are also depended on users since they need human inputs, such as voice commands and user clicks.

5.5.2 Design Elements

Each of the four main components is detailed below.

Navigation Component:

This component is used for getting location of the user and finding the path between the location of the user and the target place. For doing this, the component needs the target place. This parameter is come from voice component. The user says where s/he wants to go, and the voice recognition component converts the voice to string. The output of this component is shown on the interface and converted to voice so the voice recognition component and interface component use the outputs of this component. The requirements of this component to work are only Internet. This component uses the Google Maps API as resources.

Voice Recognition Component:

This component provides the conversion from string to voice and voice to string. This components needs for both voice and string as parameter. The voice parameter comes from the user and the string can come from both navigation and image processing class objects. The output of the component can be sent to user or can be sent to the navigation part as an input for it. This component uses the Google APIs for conversion. This component also requires the internet for conversion.

Image Processing Component:

This component is used for detecting obstacles on the yellow lines. The input of this program comes from the voice component and the output of the component goes to again voice component. However for starting object detection, the components need to wait for opening navigation component. In other words, this component requires the navigation object to be initialized. Camera is also another requirement for this component to work since the application manipulates the pictures taken from mobile phone's camera. This component uses the OpenCV libraries and functions.

Interface Component:

This component shows the output of the application. The inputs of these components are user clicks and the output of this component shown interface. In order to show the output screen the application needs outputs of navigation part and core functionalities of Android operating system.

5.5.3 Example Language

A related system package diagram is shown on the figure 7.

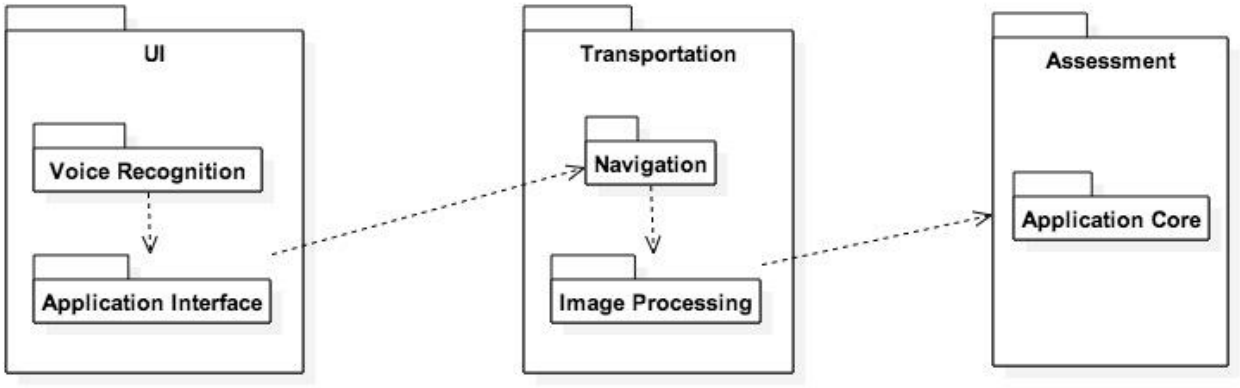

Figure 7 – Package Diagram

5.6 Interface Viewpoints

In this part of the project is provided to show user and external interfaces, and relationship between them. UML Component diagram is used for this section of the document.

5.6.1 Design Concerns

This viewpoint is used for describing the relationships between interfaces. User actions start the Comrade Application system. This starting action is done by clicking the icon of the application on the users' application menu. The interfaces are not designed for blind people. In the figure 3 of section 5.3, relationships between necessary components can be seen.

5.6.2 Design Elements

There will be one graphical user interface for the use of people who are not visually impaired. There will be four basic pages belonging to this GUI. This GUI will be responsible for providing communication between the user and Navigation Component of the application. Although the range of features dedicated to blind people is much wider and include those dedicated to non-blind people, there will be no active graphical user interface when the active user is visually impaired. If this is the case, the user interface control will be handled by voice commands and responds.

Before mentioning the specific features of each page, there are some common features of all pages in the user interface of Comrade. Firstly, all these pages will be displayed on Android

operating system. Moreover, there will be two options for the language of the pages. They will be displayed in Turkish or in English according to the user options. To change the language of the page, each page will contain two buttons; “tr” and “en”. In addition to language characteristics, each page will contain the application name (“Comrade”) on top of it. Furthermore, each page will contain a map showing the current location of the user.

Properties of each page of the application are listed below starting from the main page.

Main Page

Basic graphical user interface for main page of the application can be seen in Figure 8 in English and Turkish. In addition to map mentioned above there will be a text box that will enable the user to enter the address or name of the place to be arrived. Moreover, there will be a button (“Go” in English, “Git” in Turkish) to get the confirmation of the user about completeness of the target address. If the user does not enter a valid address or cannot be found by the application, then a pop-up screen will be displayed on top of this page to warn the user. When the user enters a valid address, he or she will be redirected to the second page of the application which is the bus page. This is the default page in between the main page and the other page.

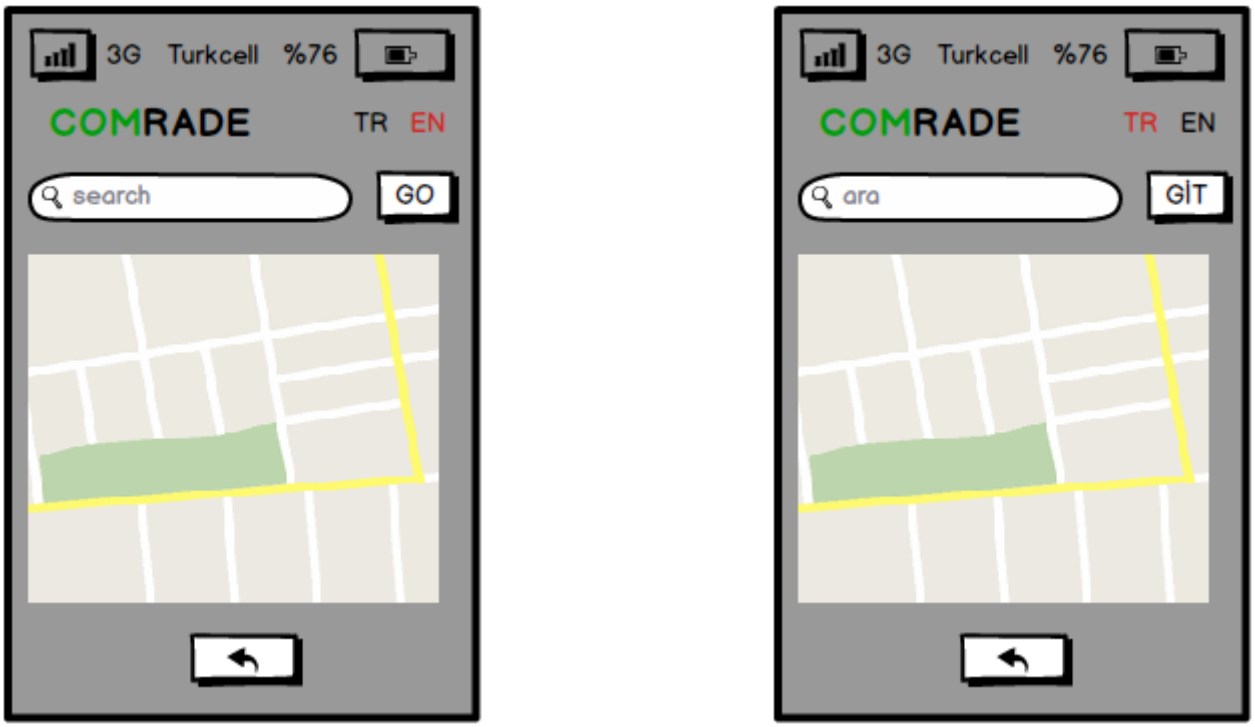

Figure 8 – Main Page

Bus Page

Basic graphical user interface for bus page of the application can be seen in Figure 9 in English and Turkish. According to specified address in main page, bus page will show the location of the most suitable bus station in the map as well as the current location of the user and a proper path between these points. Moreover, distance to this bus station from the current location will be displayed in kilometers next to the map. Lastly, there will be a dropdown list on which “Bus” in English or “Otobus” in Turkish is written. This dropdown list will be used to enable the user to change his or her options about the type of the transportation. If the user selects “Walking” (“Yuru” in Turkish) or “Taxi” (“Taksi” in Turkish) from this list, he or she will be redirected to walking page or taxi page, respectively. If the user selects “Bus” again from the list, nothing will happen and the user will stay in bus page.

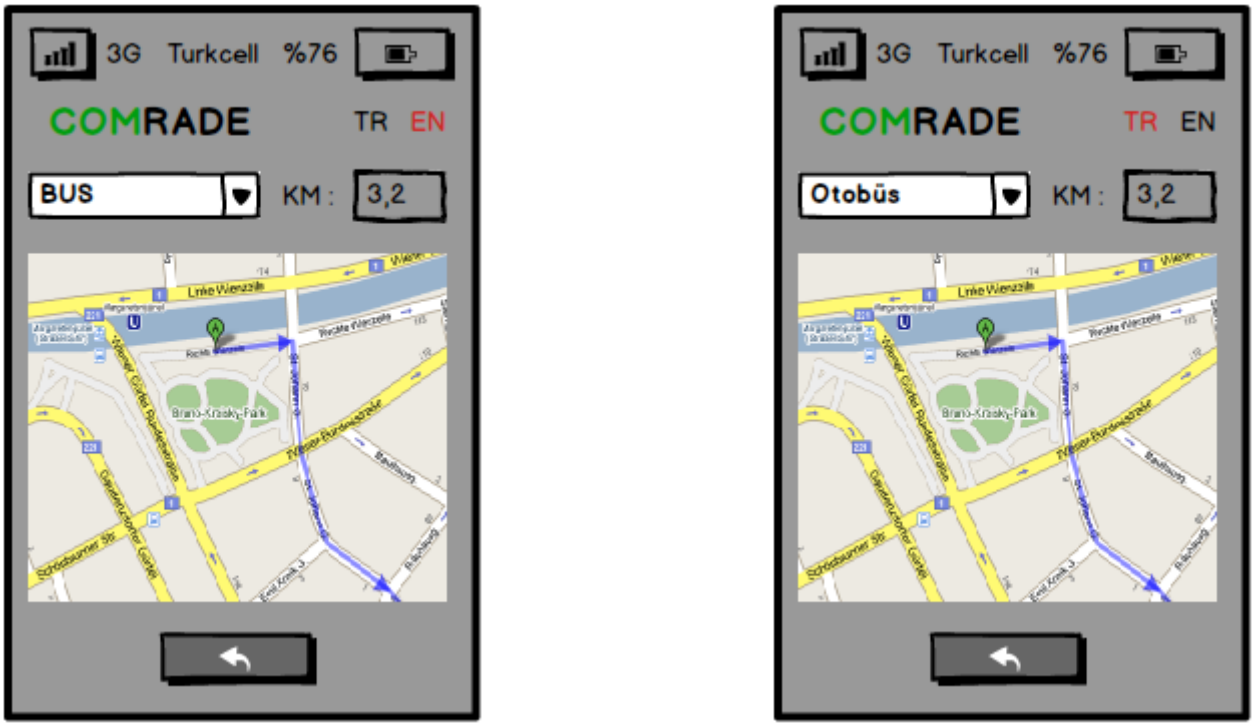

Figure 9 – Bus Page

Taxi Page

Basic graphical user interface for taxi page of the application can be seen in Figure 10 in English and Turkish. As in walking page, target location will be shown in the map as well as the current location because it is assumed that the user will call the taxi and bring the taxi to the current location of his or her. To enable this to the user, there will be a “Call” button (“Ara” in Turkish) on the page and if pushed, the application will call the closest taxi station according to current location. Similarly a path and the distance in kilometers between the current location and the target location will be displayed. As former two pages, there will be a dropdown list on which Taxi” in English or “Taksi” in Turkish is written. The aim of this drop down list is the same as the ones in Bus Page and Walking page. If the user selects “Bus” (“Otobüs” in Turkish) or “Walking” (“Yürü” in Turkish) from this list, he or she will be redirected to bus page or walking page, respectively. If the user selects “Taxi” again from the list, nothing will happen and the user will stay in taxi page.

Figure 10 - Taxi Page

Walking Page

Basic graphical user interface for walking page of the application can be seen in Figure 11 in English and Turkish. Differently from the bus page, the map inside this page will show the target location itself as well as the current location of the user. Similarly a path between the current location and the target location will also be displayed inside the map. Distance between these locations will be displayed in kilometers next to the map. As in the bus page, there will be a dropdown list on which “Walking” in English or “Yürü” in Turkish is written. The aim of this drop down list is the same as the one in Bus Page. If the user selects “Bus” (“Otobüs” in Turkish) or “Taxi” (“Taksi” in Turkish) from this list, he or she will be redirected to bus page or taxi page, respectively. If the user selects “Walking” again from the list, nothing will happen and the user will stay in walking page.

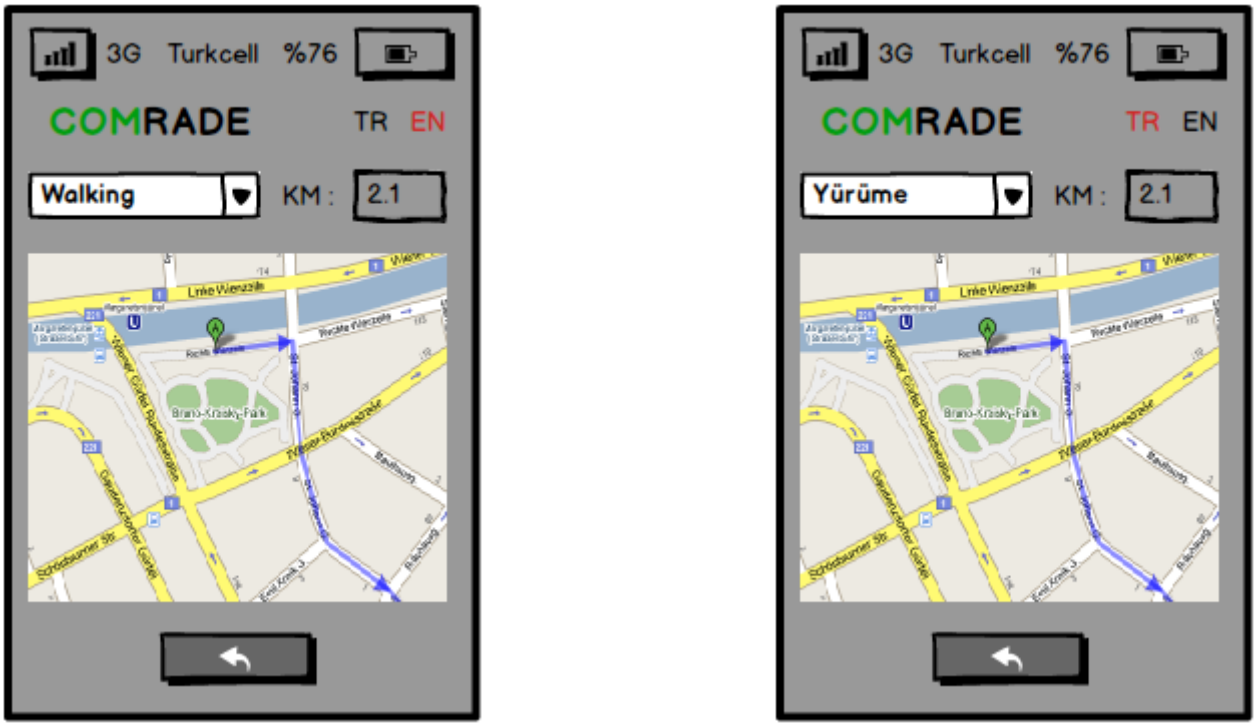

Figure 11 - WALKING PAGE

5.6.3 Example Language

In the figure 3 in section 5.3, the component diagram can be seen. The components of the application and how they interact to each other is illustrated.

5.7 Structure Viewpoints

All the necessary information about this viewpoint can be found on this document, section 5.3 composition viewpoints and section 5.5 dependency viewpoints. In those sections how the components of this system and how they interconnect and interact with each other were detailed. Moreover, for each component, its requirements, possible inputs and outputs were detailed. The necessary diagrams were also found on those sections.

5.8 Interaction Viewpoints

Interaction viewpoints are provided through UML sequence diagram, to explain the main functionalities of the Comrade Application sequences will be shown. The necessary sequence diagrams are in the section 5.8.3, example languages part.

The detailed version of use case definitions were stated in SRS document. In addition to those definitions, the general descriptions of each use cases, and corresponding sequence diagram of these use cases are described below:

- *Start Application:* This use case explains that the users of this application start the application with voice control or by clicking its icon. In figure 12, the diagram shows sequence of the start application.
- *Stop Application:* This use case explains that the users of this application can stop the object detection. In figure 13, the diagram shows sequence of the start and exit from application.

- In figure 14, main sequence diagram is shown. This is the most important sequence diagram of the application since it includes 4 of use case and actions. It shows how users select their transportation method. These use cases are descripted below:

    - *Select Method:* The select method use case explains that the users of this application can choose how they want to go desired location. The methods are walking, taxi and bus.
    - *Use Walking:* This use case explains that the users can reach the place that they want to go by walking.
    - *Use Taxi:* This use case describes that the users can reach the place that they want to go by taxi.
    - *Use Bus:* This use case defines that the users can reach the place that they want to go by bus.
- *Go to a Location:* This use case explains that the users of this application set the place or location that they want to go. In figure 15, the sequence of how the application get path, or take user to desired place is shown.
- *Start Object Detection:* This use case describes that the users of this application can be warned if there is an object on the yellow line in approximately three meters. In figure 16, the diagrams shows that how to start object or obstacle detection.

- *Stop Object Detection:* This use case describes that the users of this application can stop the object detection. In figure 17, the diagrams shows that how to stop object or obstacle detection.

- *Send SMS:* This use case explains that the users of this application send message their location. With this property, they can inform their relatives or friends about where they are, easily. Figure 18 shows the sequence of how to send SMS of location of blind people.

- *Get Battery Status:* This use case explains that the users of this application can be informed about their battery status. Figure 19 shows the sequence of how to get battery status of mobile phone.

- *Pin Location:* This use case explains that the users of this application save or pin their commonly used locations with voice control. Figure 20 shows the sequence of how to and how to pin location.

5.8.1 Design Concerns

The purpose of this is system is to explain how the application flows. The main purpose of this application is to take blind user to the place where they want to go. Also there are other important and useful functionalities of this application. All of these as well as the behavior of the system and classes is illustrated through UML sequence diagrams. Also the relationships and transitions between

There are 4 user interfaces. They are main page, taxi page, bus page and walking page. Each of them is detailed in SRS and will be detailed again in this document in section 5.6.

5.8.2 Design Elements

The application is based on user actions. For blind people these actions are voice commands. Therefore, when user starts to give commands our application parse it and calls necessary components and their functions. The voice component deals with the parsing input and output, doing conversion string to voice and vice versa. The navigation functions get the location and path, and finally the image processing component decides whether there is any object on the yellow lines. All of the outputs are converted to voice and presents the user. The application is also based on user clicks as actions of other than blind people. As a result when user clicks on the application interface, necessary components are also called according to input clicks. The only components that work with the interface is navigation so the inputs go to this component and output comes from again only this component.

5.8.3 Example Languages

The necessary sequence diagrams are as follows.

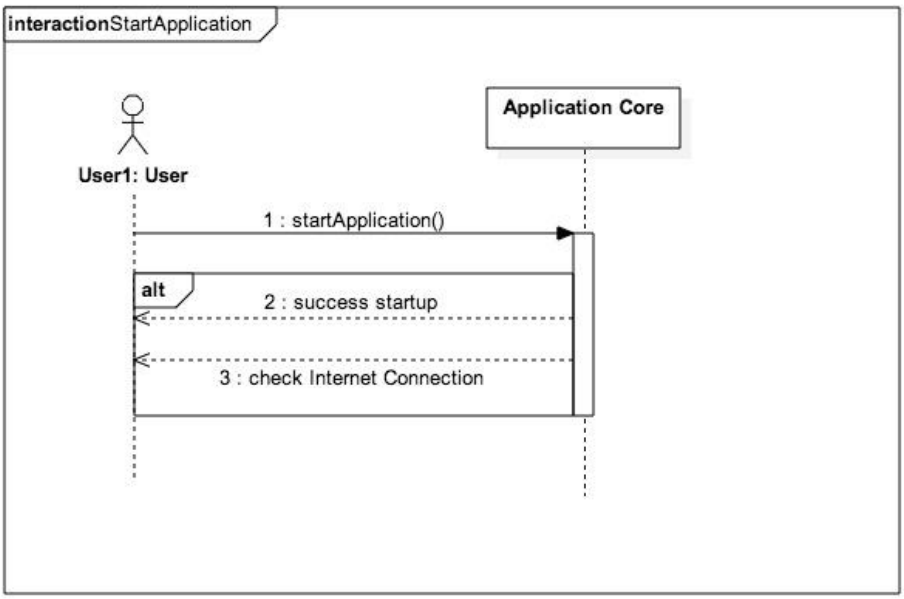

Figure 12 - START APPLICATION SEQUENCE DIAGRAM

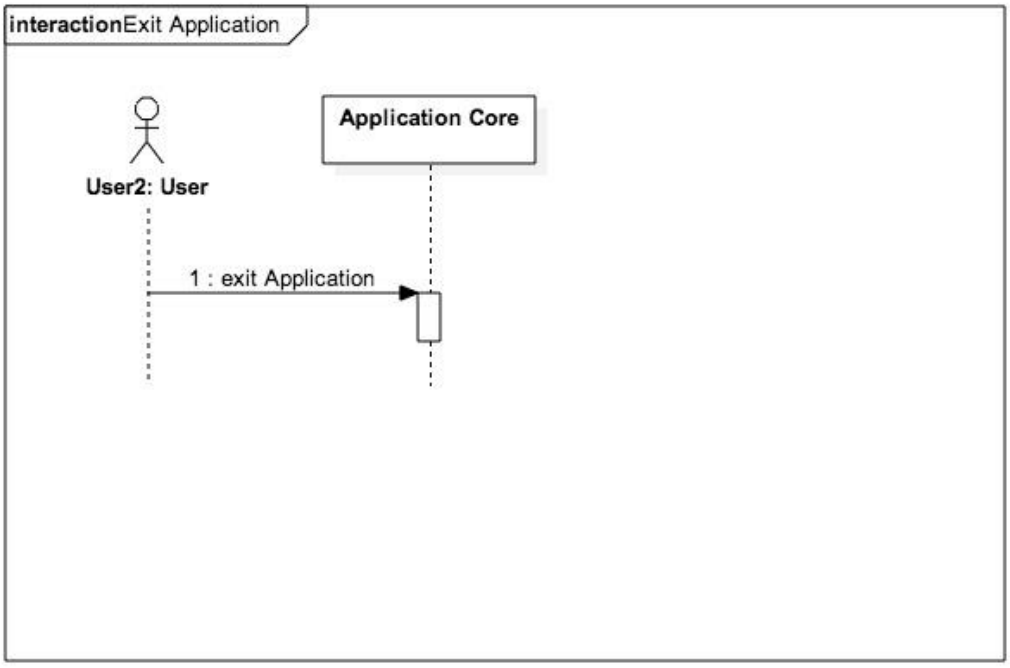

Figure 13 - EXIT APPLICATION SEQUENCE DIAGRAM

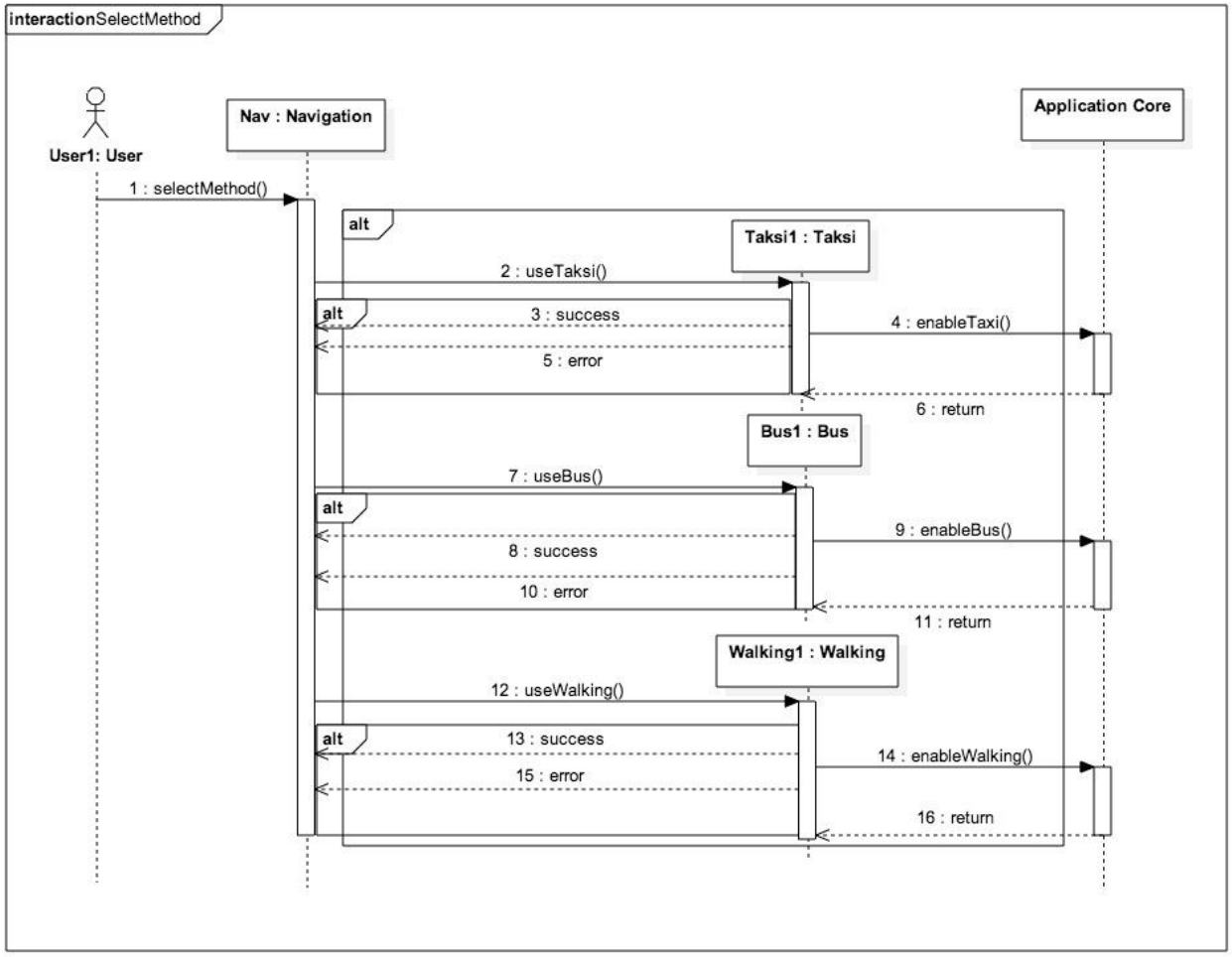

Figure 14 - MAIN SEQUENCE DIAGRAM

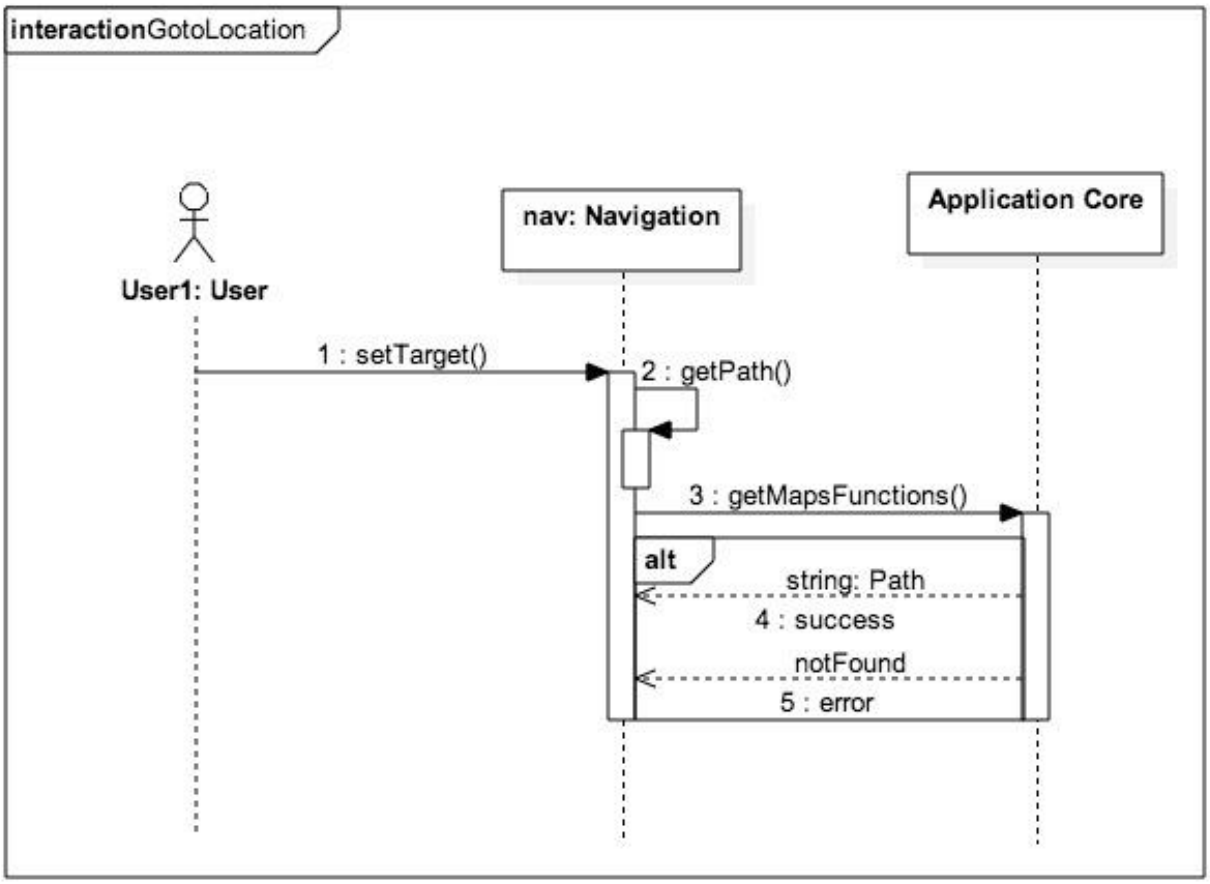

Figure 15 - GO TO A LOCATION SEQUENCE DIAGRAM

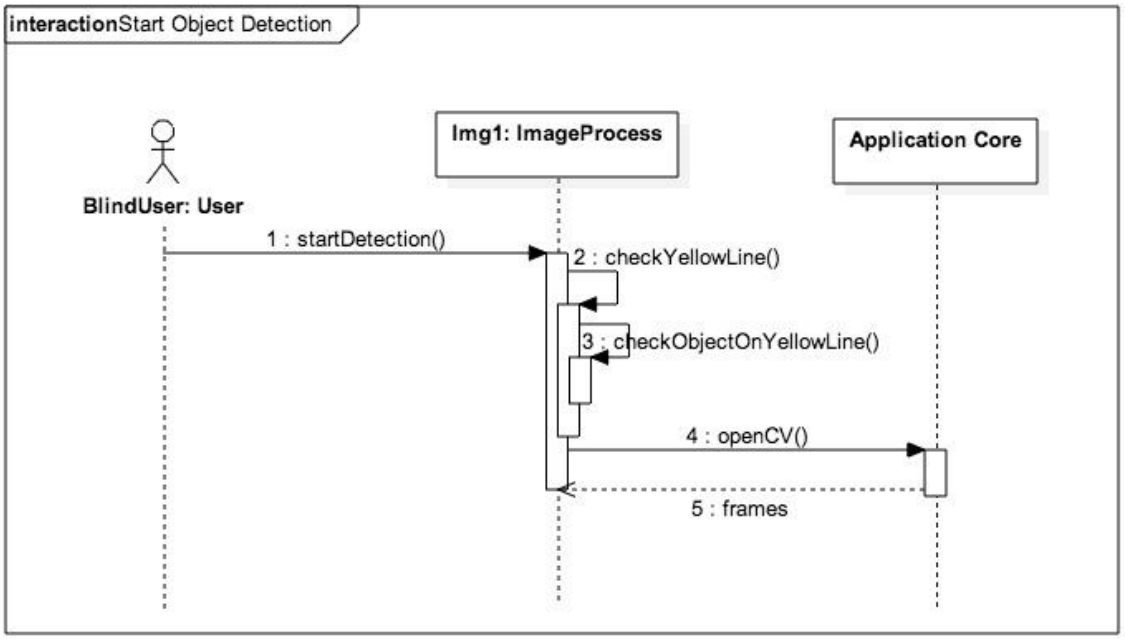

Figure 16 - START OBJECT DETECTION SEQUENCE DIAGRAM

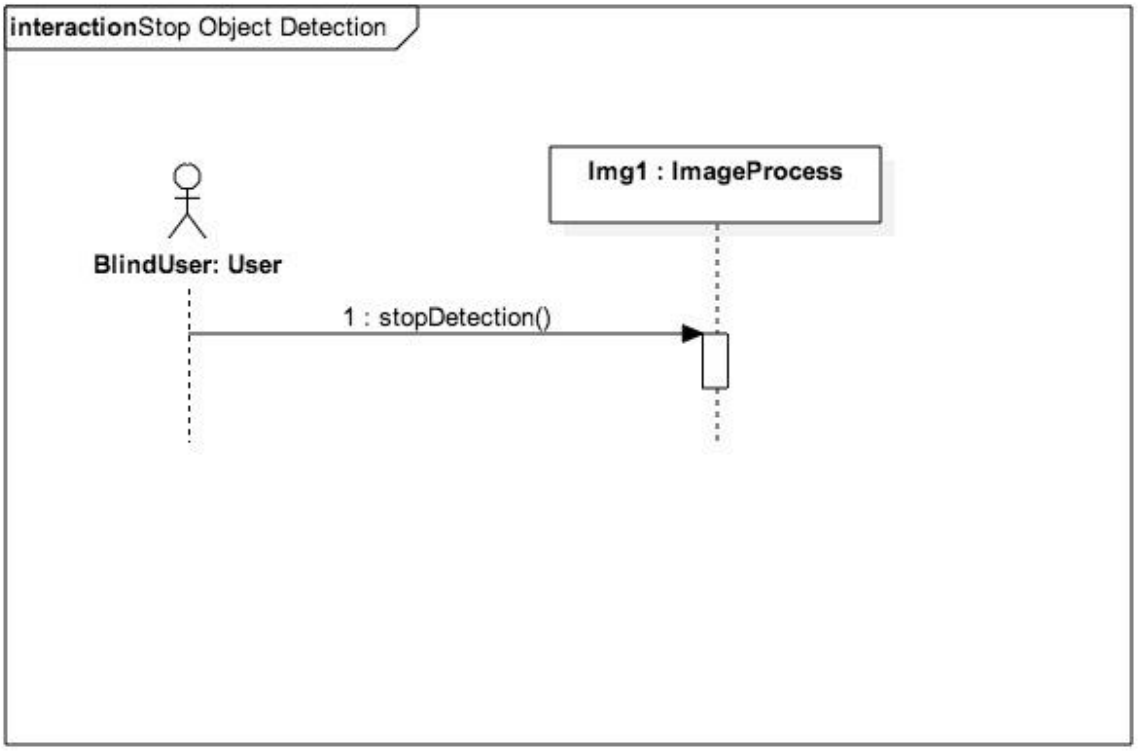

Figure 17 - STOP OBJECT DETECTION SEQUENCE DIAGRAM

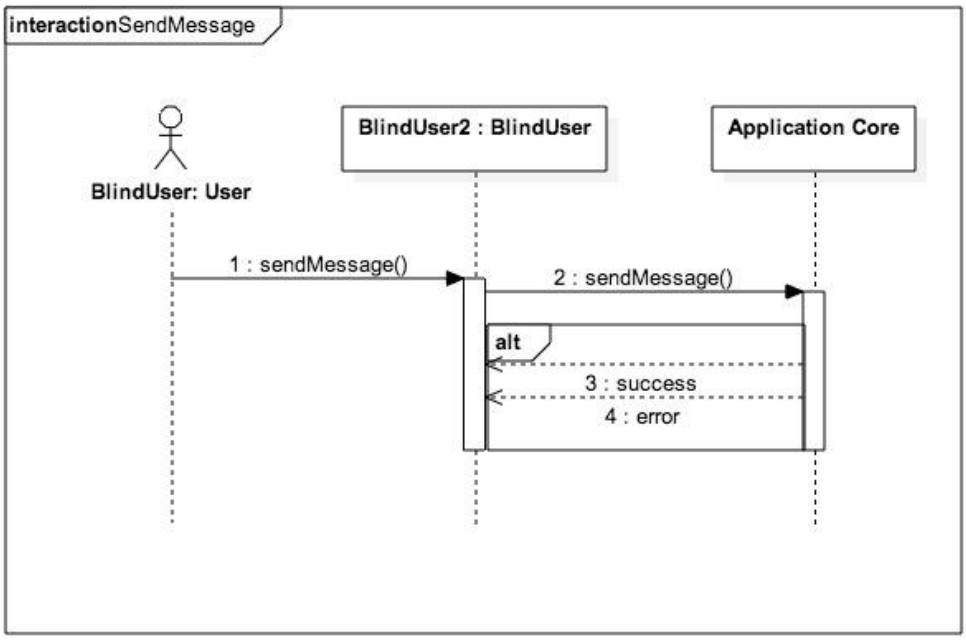

Figure 18 - SEND SMS SEQUENCE DIAGRAM

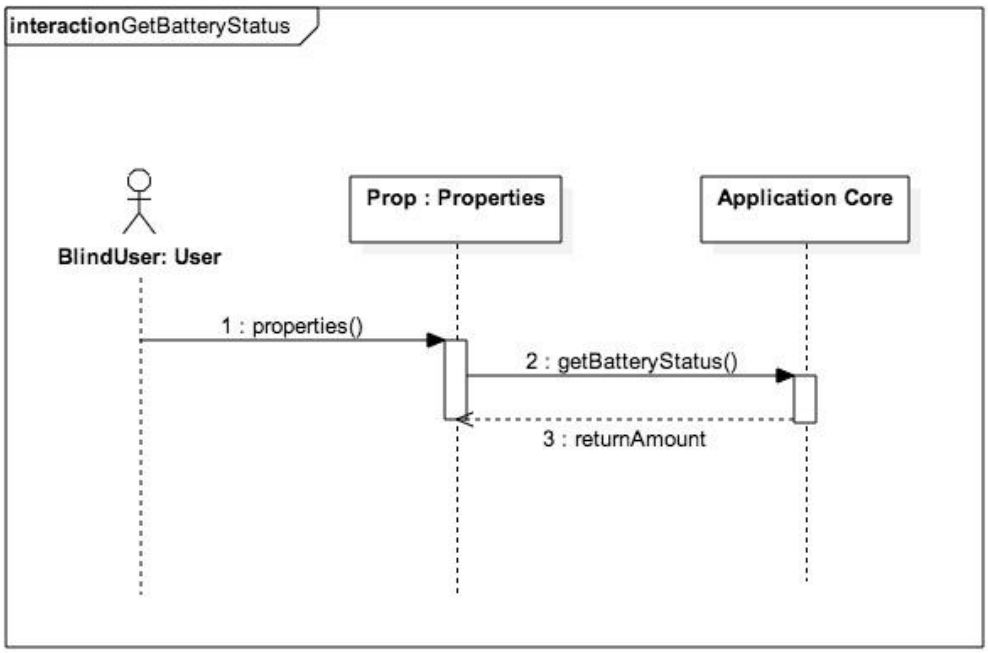

Figure 19 - GET BATTERY STATUS SEQUENCE DIAGRAM

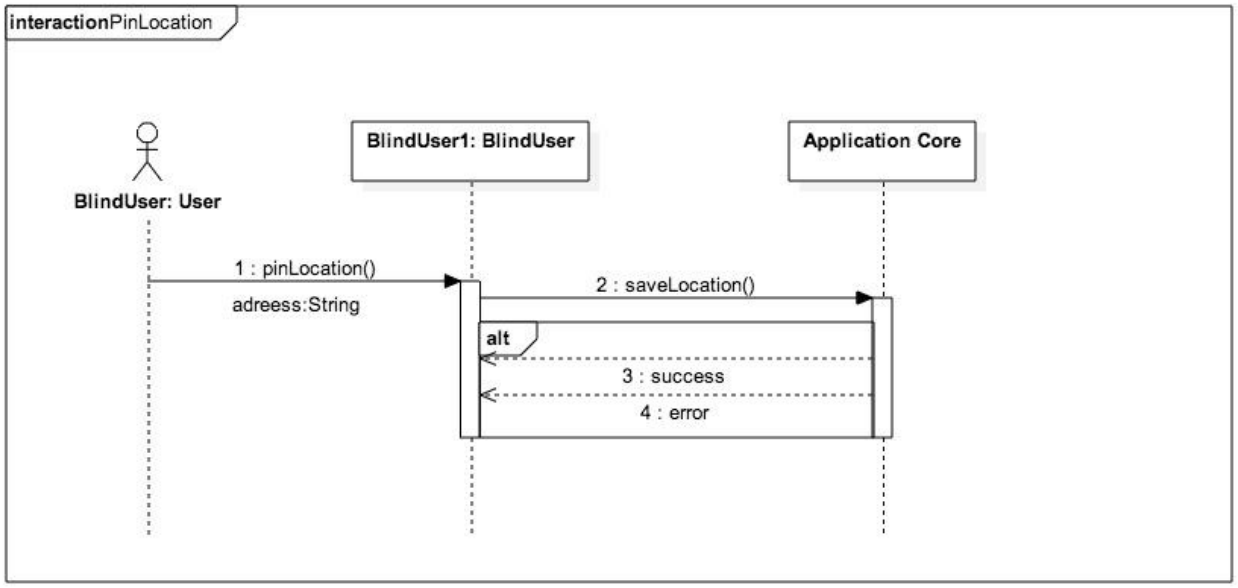

Figure 20 - PIN LOCATION SEQUENCE DIAGRAM

5.9 State Dynamics Viewpoints

The system behavior and states of Comrade Application will be detailed in this section. The example language is UML State Machine Diagram.

5.9.1 Design Concerns

Basically the application is divided into 2 separated state machines. One is for blind people and the other is for regular users, mainly tourists. For blind users, the state machine begins when the user gives the start application command and for other users the machine starts with clicking the icon of the program. The main state of the application is the initial state. At this state there are lots of options for the blind users. On the other hand for other users, there are not as many states as blind users. All states are detailed in the following section, both for blind and regular users, and diagrams are in the 5.9.3 example languages section.

5.9.2 Design Elements

The state chart diagram in figure shows all the states, triggers and conditions for each transition and related events. The trigger induces a state change given that the condition is satisfied. The conditions are written on the arrows that is takes application from one state to another state.

For Blind Users:

The first state is the main state for this application. Trigger for this state is the start application command. At this state the user has lots of options. Firstly, they can save or pin their location through voice command that is save my location. Also at this Pin Location state, after pin task is done the user go back to the main state automatically. The users also can send SMS to their location to their friends or relatives. They can send SMS only at the main state. In this main state they can learn their battery status. Since the battery status is very important they can get this information at Object detection state and navigation state. The users can start navigation part by giving the address where they want to go. After that Application goes into the select method state. At this state the user has 3 options that are bus, walking and taxi mode. They can select the method by voice commands. After that application goes into the Map state. At this state for bus, the nearest bus station is output, for taxi nearest taxi station is output and for walking target place is the output. At this state the output is converted to voice and given to the users. Moreover at this state, the users can start object detection and also stop object detection if it opened before. After the user arrive the desired location the application goes to the main state. The user can exit from the application at main state.

For Regular Users:

The first state is the main state for this application. Trigger for this state is clicking the application icon on main menu of the mobile phone. At this state the users have only one option. They can enter the place where they desire to go after that they have 3 options as in the case blind people, namely bus, taxi and walking mode. After that the application works as for blind people however the output is not voice, this time output is map shown on the screen. After the users reach the desired place, the application returns the main screen and main state.

5.9.3 Example Languages

The state chart in figure 21 and 22, the behavior of the system is illustrated. In the figure 21, the state machine for blind people is described, and in the figure 22, the state machine for regular users is illustrated.

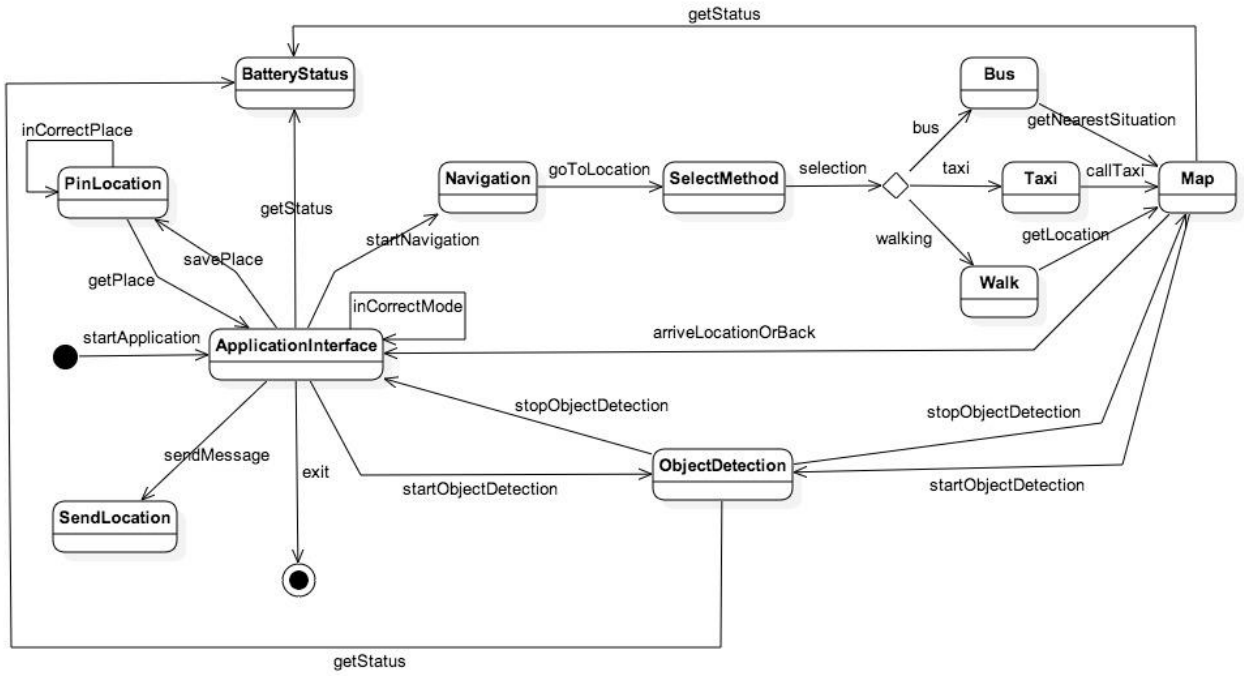

Figure 21 - State Diagram for Blind Users

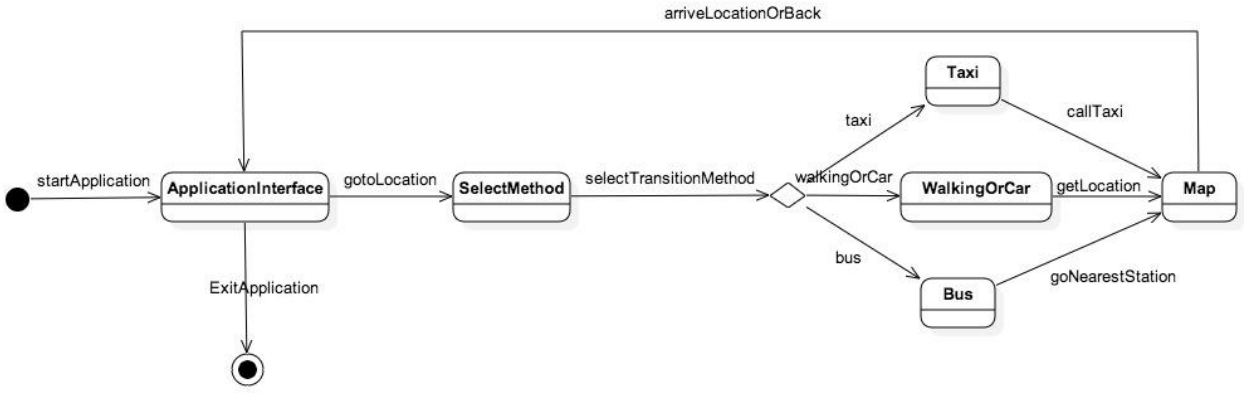

Figure 22 - State Diagram for Regular Users

6. Conclusion

This SDD is prepared to give brief information for all design patterns of Comrade Application. First, general overview and definitions of project were given. Then, relative information about design's concerns is mentioned. After that, system architecture is provided with all of its components. In the following sections, user interfaces and actions of objects are stated in design viewpoints section.