Middle East Technical University

Department of Computer Engineering

Computer Engineering Design 1

Fall 2011

Detailed Design Report for PATİKA Project

Striders of the Modern World

Levent Oral

Pınar Yılmazer

Bahar Şevket

Gözde Özcel

Table of Contents

1. Introduction

This Software Design report is prepared by “SMW” group members to determine 3D Map Visualization and Path Finding project requirements. It is designed to help readers to understand the concepts of the project and use this tool easily and effectively. An overview of the problem and the product is described. Design requirements, data design and system architecture are also introduced.

1.1 Problem Definition

Military logistics is a process of planning and carrying out the movement and maintenance of military forces. Therefore, fuel delivery for military vehicles has an important place in military logistics and it must be handled in the most efficient way. Today, fuel delivery systems are planned with the use of different type of maps and software applications. While vector maps, which are most widely used maps, contain features like coordinates, major road networks, railroad networks and international boundaries, they do not contain the height data of a point. However, military vehicles are capable of moving on not only the human made roads, but also on different types of terrains. As a result, while calculating shortest distance and shortest trip time for fuel delivery purposes, the topographic map, maps showing height of a particular area, must be used. Moreover, the vehicle`s capabilities, such as moving on different terrain types, moving on a slope and fuel amount, should be considered. Although there are many applications offering shortest path finding in the market, there is no such application that offers all the capabilities of this project aims to offer. Thus, in this project these issues will be addressed.

1.2 Purpose

This Software Design Document is written in order to provide a complete description of all the aspects of 3D Map Visualization and Path Finding project in order to give the developers a guidance of the architecture of the software. It introduces detailed functionalities, use case, class, data flow, component, sequence diagrams, constraints and dependencies of the project. It also gives information about user interface. 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. Because this project is intended to be used for military purposes, our user profile is officers responsible for military logistics.

1.3 Scope

This document contains a complete description of the design of 3D map visualization and path finding for fuel delivery purposes, with different terrains and characteristic features of vehicles for Android. All the components and their functions are explained in the document.

1.4 Overview

The following chapters and their contents are

- Part 2 is System Overview that includes a general description of the overall system and its design. The differences of this product from similar products in the market are explained in this part.
- Part 3 is Design Considerations. This part briefly introduces issues related to design, constraints that affect the design and the architecture of the software and any design goals and principles that form the software.
- Part 4 is the Data Design that lists how the information domain of our system is transformed into data structures, and also, it lists how the data entities are stored, managed, or manipulated with their descriptions, types and attributes.
- Part 5 is the System Architecture. This is the most important part of the document. It specifies the design entities that collaborate to perform the functionality of the system. Each of these entities presents the features that it will provide to the system. To clearly explain the design, a component diagram for the software is drawn and all the packages and their classes are shown in the package diagrams.
- Part 6 explains the contents of the user interface. It shows graphical user interface of the application and clarifies how each screen object functions. The screenshots of the user interface are also included in this chapter.
- Part 7 lists the libraries and tools that will be used in development process.
- Part 8 includes Gantt chart illustrating the planning of the project, spanning two separate semesters.
- Part 9 is the conclusion.

1.5 Definitions and Abbreviations

SMW: Striders of the Modern World

SRS: Software Requirements Specification

PC: Personal Computer

CPU: Central Processing Unit

DTED: Digital Terrain Elevation Data

WIFI: Wireless Fidelity

PATIKA: The name given to the 3D map visualization and path finding for fuel delivery purposes, with different terrains and characteristic features of vehicles for Android project.

1.6 References

[1]https://cow.ceng.metu.edu.tr/Courses/?semester=20111&course=ceng491&cedit=0
SDD template from the course's website

[2] IEEE Std 830-1998: IEEE Recommended Practice for Software Requirements Specifications
A common software engineering standard to provide some guidance and recommended approaches for specifying software design descriptions

[3]http://www.gliffy.com
Web-based diagram editor

[4]http://developer.android.com/guide/topics/graphics/opengl.html
Web page that usage of OpenGL ES and android is clearly explained.

[5]http://en.wikipedia.org/wiki/Use_case_diagram
Use case diagrams

[6]http://developer.android.com/guide/topics/ui/index.html
Fundamentals of creating a user interface with Android

2. System Overview

PATIKA is a military application designed for 3D map visualization and path finding for fuel delivery purposes, with different terrains and characteristic features of vehicles. This application is capable of plotting scenarios by finding the shortest delivery routes between multiple fuel delivery vehicles and vehicles that need refueling. The goals of this application will be

- To provide an accurate and efficient mobilization of military units (for refueling).
- To provide a user interface that is simple but potent, and a scenario simulator to utilize the

path finding algorithm most suitably for military logistics (refueling).

Our application will work on mobile Android platforms. Similar applications use less comprehensive path finding algorithms that do not consider the extensive cases our algorithm will have to take into account in order to be of value in military conditions.

3. Design Considerations

3.1 Constraints, Assumptions and Dependencies

The Patika Application will follow the following design constraints:

- Programming language that will be used during the project is Java, since we will be developing this application for Android platforms and Java is the native language of Android.
- We will be using Android SDK API Level 8 for this application since it is more widely supported by the mobile devices.
- The system shall run on mobile devices, so the processor speed and the quality of the graphics card should be enough to be able to run the application.
- Since the system will run on mobile devices, display size is a consideration for us. We will be supporting phone size and tablet size screen resolutions.
- The main system will run on a single machine and it will not be dependent on a bigger system or any other computers, but sending the prepared scenarios to the fuel delivering vehicles require some connection between the commanding device and the task device. For this case, we will assume secure WiFi connection is present and established, all the devices and the vehicles are matched, and the commanding device knows the IP's of all corresponding vehicles in the prepared scenario. This assumption is due to the fact that military intelligence systems use radio transmissions to send / receive data, and we do not have the sufficient equipment to simulate that transmission environment.
- Since we will need wide ranges for visualizing long routes of the fuel delivery vehicles, our application should not wait too much to render the new areas of the map, and if an error occurs, user shall be notified immediately.
- For visualization purposes, support for simulation of land vehicles and air vehicles in the same scenario will not be given.
- To get the maximum visual quality from our application and to conform to the real world standards of maps (width/height ratio is generally bigger than 1), we will be

displaying our application only on landscape mode (horizontal position) on the mobile devices.

- We will assume infinite fuel in the tanks of carrier vehicles to reduce complex calculations on fuel consumption.
- We will be supporting two types of users. First one is the commanding officer type user who will have full access to the features of the system. Second is the task force type user who will have restricted access to the features of system. These restrictions will be put to the system from the beginning (different apk).

3.2 Design Goals and Guidelines

An important design goal is to keep the graphical user interface simple. Users shall not grapple with the complex structure and flow of events. User interface shall be smooth and almost linear, taking the user from the beginning to the end with subtle directing.

Also, accuracy of the system is important for the users. For this purpose, path finding algorithm will consider many factors that can affect the accuracy of the generated paths.

To keep the task device's software secure, we will disable control functionalities of the commanding (original) software, in other words, users of the task software will only be able to play the scenario that was sent to them.

4. Data Design

4.1 Data Description

We have no database in our application. However, there are some raw data that needs storage. This is accomplished using simple XML files.

- Files of vehicles: These files store the detailed features of vehicles for all the models available. Vehicle classes in Model Package (section 5.2.1) will be constructed using this data.
- Files of maps: These files store the DTED2 representation of the available maps. Map class in Model Package (section 5.2.1) will be constructed using this data.
- Files of air corridors: These files store the coordinates and sizes of the various air corridors available. AirCorridor class in Model Package (5.2.1) will be constructed using this data.

Other than XML files, sending generated scenarios to task computers is an important data manipulation. Connections package in 5.2.5 handles the connecting and sending operations required for this feature.

4.2 Data Dictionary

4.2.1 Model Package

4.2.1.1 Class BasicVehicle

Attributes:

- **String model:** This field keeps the model of the vehicle and can be used to verify correct vehicle type, chosen by the user, within a range of models, usually of different sizes and capabilities. This field cannot be modified, therefore; it will be read from raw data file (vehicle) when BasicVehicle class is constructed. User can only see the model of the vehicle when he/she wants to see detailed display of features.
- **Interger IP:** This attribute keeps the Internet Protocol address of the vehicle. It is used to send simulation data to the right vehicle at the simulation tab of the GUI. This field can not be modified, so must be set initially. User can only see the IP of vehicle when he/she wants to see detailed display of features.
- **Integer meshIndex:** This field keeps the mesh index for vehicle. When the map is meshed according to constraits of the relevant vehicle, meshIndex is set in order to determine the relevant meshed maps and paths for this vehicle.
- **Integer maxPassenger:** The maximum passenger count that a vehicle can carry is kept in this field. This field can not be modified, so must be set initially.
- **Integer avePassenger:** The current number of passengers that a vehicle carries, is kept in this field. This field can be modified, so it will be set each time.
- **Float fuelAmount:** The current fuel carried by the vehicle is kept in this field. For carrier vehicles, this field shows current fuel amount that carrier vehicle can supply to receiver vehicle. For receiver vehicles, this field shows remaining fuel. This field can be set each time by the user, when user creates a new scenerio.
- **Float fuelCapacity:** The maximum fuel capacity of the vehicle is kept in this field. If mentioned vehicle is carrier type, this field shows maximum amount of fuel that carrier vehicle can supply to receiver vehicles. If mentioned vehicle is receiver type, this field shows maximum possible amount of fuel storage.

- **Float aveSpeed:** This attribute keeps the average speed of the vehicle. This field cannot be modified, therefore; it must be set initially when vehicle class is constructed.
- **Float height:** The height of the vehicle is kept in this field. This field cannot be modified, therefore; it must be set initially when vehicle class is constructed. User can only see the height of a vehicle when he/she wants to see detailed display of features.
- **Float width:** The width of the vehicle is kept in this field. This field can not be modified, therefore ; it must be setted it initially when vehicle class is constructed. User can only see the width of a vehicle when he/she want to see detail display of the relevent vehicle.
- **Float weight:** The weight of the vehicle is kept in this field. This field cannot be modified, therefore; it must be set initially when vehicle class is constructed. User can only see the weight of a vehicle when he/she wants to see detailed display of the features.
- **Point coordinates:** This field keeps the information of where the vehicle is, in terms of latitute, longitute and altitude. This field can be set each time by the user, when user creates a new scenerio.

Methods:

- **void setModel(String):** This method takes a string as an argument and sets the model field with this argument.
- **String getModel():** This method returns the model field, and it does not take any arguments.
- **void setIP(Integer):** This method takes an integer as an argument and sets IP field with this argument.
- **Integer getIP() :** This method returns the IP field, and it does not take any arguments.
- **void setMeshIndex(Integer) :** This method takes an integer as an argument and sets the meshIndex field with this argument.
- **Integer getMeshIndex():** This method returns the meshIndex field, and it does not take any arguments.
- **void setMaxPassenger(Integer)**: This method takes an integer as an argument and sets the maxPassenger field with this argument.
- **Integer getMaxPassenger():** This method returns the maxPassenger field, and it does not take any arguments.
- **void setAvePassenger(Integer) :** This method takes an integer as an argument and sets the AvePassenger field with this argument.

- **Integer getAvePassenger() :** This method returns the avePassenger field, and it does not take any arguments.
- **void setFuelCapacity(Float) :** This method takes a float as an argument and sets the fuelCapacity field with this argument.
- **Float getFuelCapacity():** This method returns the fuelCapacity field, and it does not take any arguments.
- **void setFuelAmount(Float) :** This method takes a float as an argument and sets the fuelAmount field with this argument.
- **Float getFuelAmount() :** This method returns the fuelAmount field, and it does not take any arguments.
- **void setAveSpeed(Float):** This method takes a float as an argument and sets the aveSpeed field with this argument.
- **Float getAveSpeed():** This method returns the aveSpeed field, and it does not take any arguments.
- **void setHeight(Float):** This method takes a float as an argument and sets the height field with this argument.
- **Float getHeight():** This method returns the height field, and it does not take any arguments.
- **void setWidth(Float):** This method takes a float as an argument and sets the width field with this argument.
- **Float getWidth():** This method returns the width field, and it does not take any arguments.
- **void setWeight(Float):** This method takes a float as an argument and sets the weight field with this argument.
- **Float getWeight():** This method returns the weight field, and it does not take any arguments.
- **void setPointCoordinates(Point):** This method takes a Point object as an argument and sets the pointCoordinates field with this argument.
- **Point getPointCoordinates():** This method returns the pointCoordinates field, and it does not take any arguments.

4.2.1.2 Class LandVehicle

LandVehicle class extends BasicVehicle class.

Attributes:

- **Float maxSlope:** This field keeps the maximum incline that land vehicle can keep moving unproblematically. This field can not be modified, therefore; it must be set initially when LandVehicle class is constructed. The user can only see the maximum slope attribute when he/she wants to see detailed display of features.
- **Float minSlope:** This fields keeps the minimum maximum incline that land vehicle can keep moving unproblematically. This field can not be modified, therefore; it must be sett initially when LandVehicle class is constructed. The user can only see the minimum slope attribute when he/she wants to see detailed display of features.
- **Boolean waterRes:** This attribute keeps the information of whether a land vehicle is able to move in the water or not. This field can not be modified, therefore; it must be set initially when LandVehicle class is constructed.
- **Float maxWaterDept:** This attribute keeps the information of maximum water depth a vehicle can move if its waterRes attribute is true. This field can not be modified, therefore; it must be set initially when LandVehicle class is constructed.

Methods:

- **void setMaxSlope(Float)**: This method takes a float as an argument and sets the maxSlope field with this argument.
- **Float getMaxSlope() :** This method returns the maxSlope field, and it does not take any arguments.
- **void setMinSlope(Float) :** This method takes a float as an argument and sets minSlope field with this argument.
- **Float getMinSlope() :** This method returns the minSlope field, and it does not take any arguments.
- **void setWaterRes(Boolean) :** This method takes a boolean as an argument and sets the waterRes field with this argument.
- **Boolean getWaterRes() :** This method returns the waterRes field, and it does not take any arguments.
- **void setMaxWaterDept(Float) :** This method takes a float as an argument and sets the maxWaterDept field with this argument.
- **Float getMaxWaterDept() :** This method returns the maxWaterDept field, and it does not take any arguments.

4.2.1.3 Class AirVehicle

AirVehicle class extends BasicVehicle class.

Attributes:

- **Float maxAltitude:** This field keeps the maximum altitude information that an air vehicle can fly at. This field can not be modified, therefore; it must be set initially when AirVehicle class is constructed. User can see the maximum altitude attribute only when he/she wants to see detailed display of features.
- **Float minAltitude:** This fields keeps the minimum altitude information that an air vehicle can fly at. This field can not be modified, therefore; it must be set initially when Airvehicle class is constructed. User can only see the minimum altitude attribute when he/she wants to see detailed display of features.

Methods:

- **void setMaxAltitude(Float):** This method takes a float as an argument and sets the maxAltitude field with this argument.
- **Float getMaxAltitude():** This method returns the maxAltitude field, and it does not take any arguments.
- **void setMinAltitude(Float):** This method takes a float as an argument and sets minAltitude field with this argument.
- **Float getMinAltitude() :** This method returns the minAltitude field, and it does not take any arguments.

4.2.1.4 Class Path

Attributes:

- **BasicVehicle _from:** This field keeps the vehicle object that is responsible from supplying fuel.
- **BasicVehicle[ ] _to:** This field keeps an vehicle object array list whose members are the receiver vehicles waiting for refuelling.
- **Float[ ] intervalAveTime** : This field keeps the interval average time array list. Each member of list shows the time passed from starting point to the ending point of the corresponding sub path.
- **Float [ ] AveDistance :** This field keeps the distance array list. Each member of list shows the distance calculated from starting point to the ending point of the corresponding sub path.

- **Point[ ] pathPoints :** This field keeps the point object array list. Each member of list shows the corresponding points which are sampled from main path periodically.

Methods:

- **void set_from(BasicVehicle) :** This method takes a BasicVehicle object as an argument and sets the _from field with this argument.
- **BasicVehicle get_from() :** This method returns the _from field, and it does not take any arguments.
- **void insert_to(BasicVehicle) :** This method takes a Vehicle list as an argument and inserts it to _to field.
- **BasicVehicle [ ] get_toByIndex (int index) :** This method returns a BasicVehicle object according to the index and the _to field.
- **void calculateT() :** This method calculates average travel time for each sub path according to starting and ending point of the path (point information) and averageSpeed attribute of the carrier vehicle. It constructs intervalAveTime list sequentially.
- **void calculateD() :** This method calculates average distance that carrier tanker should go for each sub path according to starting and ending point of the path (point information). It constructs AveDistance list sequentially.
- **void calculateP(Float interval) :** This method calculates all points, that carrier vehicle should pass while following its main path, according to given interval value. It constructs pathPoints list sequentially.

4.2.1.5 Class Point

Attributes:

- **Vector3 height**: Height of the point object is kept in this field in Vector3 type.
- **Veccor3 latitute:** Latitude of the point object is kept in this field in Vector3 type.
- **Vector3 longitute:** Longitute of the point object is kept in this field in Vector3 type.
- **Vector3 normal:** Normal of the point object is kept in this field in Vector3 type and it is used to give smooth appearances while drawing maps.

Methods:

- **void setHeight(Vector3) :** This method takes a Vector3 object as an argument and sets the height field with this argument.
- **Vector3 getHeight() :** This method returns the height field, and it does not take any arguments.

- **void setLongitude(Vector3) :** This method takes a Vector3 object as an argument and sets longitude field with this argument.
- **Vector3 getLongitude() :** This method returns the longitude, and it does not take any arguments.
- **void setLatitude(Vector3) :** This method takes a Vector3 object as an argument and sets the latitude field with this argument.
- **Vector3 getLatitude() :** This method returns the latitude field, and it does not take any arguments.
- **void setNormal(Vector3) :** This method takes a Vector3 object as an argument and sets the normal field with this argument.
- **Vector3 getNormal() :** This method returns the normal field, and it does not take any arguments.

4.2.1.6 Class AirCorridor

Attributes:

- **Point[ ] airCorridorPoint :** This field is an array list which keeps all of the point information that is necesaary for constructing air corridor volume.
- **Point[ ] begin:** This field is an array list which keeps all of the point information that is necessary for constructing the entrance area of air corridors.
- **Point [ ] end :** This field is an array list which keeps all of the point information that is necessary for constructing the exit area of air corridors.

Methods:

- **void setAirCorridorPoint(Point [ ]) :** This method takes a point array list as an argument and sets the airCorridorPoint field with this argument.
- **Point[ ] getAirCorridorPoint () :** This method returns the airCorridorPoint field, and it does not take any arguments.
- **void setBegin (Point [ ]) :** This method takes a point array list object as an argument and sets begin field with this argument.
- **Point[ ] getBegin() :** This method returns the beginning, and it does not take any arguments.
- **void setEnd(Point [ ]) :** This method takes a point array list as an argument and sets the end field with this argument.
- **Point[ ] getEnd() :** This method returns the end field, and it does not take any argument.

- **Point calculateAveBegin() :** This method works on the begin list and calculates a proper and safe entrance point for air vehicles.
- **Point calculateAveEnd() :** This method works on the end list and calculates a proper and safe exit point for air vehicles.

4.2.2 Engine Package

4.2.2.1 Class PatikaEngine

Attributes:

- **Map map:** This attribute keeps coordinate information about the map.
- **BasicVehicle[] allVehicles:** All vehicles that user can select amongst, are stored in this attribute.
- **BasicVehicle[] selectedVehicles:** The vehicles which are currently selected are stored in this field.
- **Path[] paths:** The paths corresponding to each vehicle having type carrier are stored in paths attribute.
- **Mesh[] meshes:** The meshes that will correspond to the areas that a specified type of a vehicle can move on are stored in this field.

Methods:

- **void createMap(String path):** This method takes a String as an argument, which corresponds to the path at which DTED2 file is stored and, calls MapFactory.read() to generate the Map attribute.
- **void fillAllVehicles():** This method will call VehicleFactory to fill allVehicles attribute.
- **void addVehicleToSelected(int modelNo):** This method takes modelNo as an argument and searches for the vehicles having that modelNo in allVehicles list. If found, it adds the vehicle to selectedVehicles.
- **Vehicle[] filterVehicles(int type):** This method takes type as an argument and returns a vehicle array containing only the vehicles of a specific type.
- **void generatePaths(Map map,Mesh[] meshes, BasicVehicle[] selectedVehicles):** This method fills paths array by calling pathFinder component's methods. During the path generation process the arguments of type Map, Mesh[] and BasicVehicle[] will be needed.
- **void addMesh(Map map,BasicVehicle vehicle):** This method will add new meshes to meshes array. The corresponding mesh will be generated according to map and vehicle

arguments of the method.

4.2.2.2 Class Mesh

Attributes:

- **Polygon[] polygons:** Polygons that form a mesh

Methods:

- **void generateMesh(Map map, BasicVehicle vehicle):** This method will generate a mesh using a map and vehicle arguments.

4.2.3 Factories Package

4.2.3.1 Class MapFactory

Attributes:

- **Map map:** This attribute keeps coordinate information about map.
- **String path:** This attribute keeps the path of the DTED2 file which contains necessary information to create Map attribute.

Methods:

- **void read():** This method will read DTED2 files and creates a Map object using the information stored in the file.

4.2.3.2 Class PathFactory

Attributes:

- **Path[] paths:** This attribute contains data of paths.

Methods:

- **void parse(string pathStr):** This method will parse its argument pathStr and as a result will fill paths attribute. This method will be used by a second type of user (task device) who will only see the path sent by commander (command device).

4.2.3.3 VehicleFactory

Attributes:

- **BasivVehicle[] vehicles:** This attribute contains data of vehicles.

Methods:

- **void read():** This method reads data from vehicle XML files which store information on vehicles and fills vehicles array.
- **void write(BasicVehicle[] vehicles):** This method takes a BasicVehicle array as an argument and writes the elements of this array to an XML file.

4.2.4 Connections Package

4.2.4.1 Server

Attributes

- **ArrayList<String> ids:** This field keeps id list of connected devices to Command Device through XML Socket which are types of Mission Device.
- **ArrayList<ServerSocket> connections:** This field keeps Socket connection objects of connected devices through XML Socket to Command Device.
- **final int port:** This field determines port number which server socket is bounded to.
- **ServerSocket server:** It is socket server of class which creates a server socket bound to port.

Methods

- **void run():** This method overrides run method of Thread class. It waits Socket connections and accepts them in infinite loop. It also maps created connections to the id of connected device.
- **ArrayList<String> getID():** This method return id list.
- **void send(String id, String message):** This method sends message to specific device which is defined by its id. Message will be in XML data format which represent Path data model.
- **void close(String id):** This closes the socket connection of device specified by its id. This also removes from connections and ids list.

4.2.4.2 Connector Class

Attributes

- **String host:** Specify address of host that device try to connect to. This field has a pre-defined value which is address of specific Command Device that is identified as default.
- **int port:** Specify port number of host that device try to connect to on this port. It is initialized by default.

Method

- **void setHost(String host):** Sets the host attribute with a new value to make a custom specification on Command Device address which is target of connection.
- **void setPort(int port):** Sets the port attribute with a new value.
- **connect():** Establishes a connection with the Command Device that is defined by its host address and port.
- **run():** This method overrides run method of Thread class. It waits the message from server that is connected Command Device through Socket.

4.2.5 GUI Package

4.2.5.1 MainTab

MainTab is a manager that manages all sub-menus and their interaction according to flow of program.

Method

- **openMapMenu():** When user clicks Map Tab, manager will open Map sub-menu inside Main Tab main menu. Manager also disables all other Tabs to prevent occurrence of wrong flow in program. This restriction aims to create easier flow controller for both user and program.
- **openTankersMenu():** When user clicks Tankers Tab, manager will open Tankers sub-menu inside Main Tab main menu. Manager only disables Receiver and Simulation tabs.
- **openReceiversMenu():** When user clicks Receivers Tab, manager will open Tankers sub-menu inside Main Tab main menu. Manager only disables Simulation tab.
- **openSimulationMenu():** When user clicks Simulation Tab, manager will open Simulation sub-menu inside Main Tab main menu. Manager enables all tabs.

4.2.5.2 Map

Map sub-menu includes Maps options available in program. It shows all maps to get selection of user.

Method

- **selectMap(int mapID):** When user selects a map by double clicking on list, selected map will be rendered on the remaining sub-menus. This automatically opens Tankers sub-menu by calling reletad function from MainTab.
- **list():** Lists all maps in the program as GroupList view.

4.2.5.3 Tankers

Tanker sub-menu includes a tanker list which enables to user select one or more vehicles, update some necessity fields of vehicles. It coordinates tankers in map by getting coordinates from user.

Method

- **list():** Lists all tanker type vehicles in the program as List with disabling some features of vehicles.
- **listFilter(Vehicle[] vehicles):** Lists only filtered vehicles. Filter is made by user first choice of type of vehicle. If user selects air type tanker then only air type vehicles will be filtered to show. Moreover, if user selects land type tanker then only land type vehicles will be filtered to show.
- **select(int vehicleID):** Selects a vehicle by its index in List, when user selects a vehicle in list by double clicking on it. It also calls filtering function from PatikaEngine and show filtered list by calling listFilter function.
- **update(float longitude, float latitude, float altitude, float fuelAmount):** Updates the selected vehicle's related areas by calling PatikaEngine functions and show users updated version.
- **home():** Opens Map sub-menu in MainTab main menu by calling related function from MainTab.
- **back():** Opens Map sub-menu in MainTab main menu by calling related function from MainTab.
- **next():** Opens Receivers sub-menu in MainTab main menu by calling related function from MainTab. If no vehicle is selected, this function is disabled to prevent wrong flow of program. If vehicles are selected, this function is updates selectedVehicles in PatikaEngine by calling related update function.

4.2.5.4. Receivers

Receivers sub-menu includes a receiver list which enables to user select one or more vehicles, update some necessity fields of vehicles. It coordinates receivers in map by getting

coordinates from user.

Method

- **list():** Lists all receiver type vehicles in the program as List with disabling some features of vehicles. It is affected by filter action according to selected tanker vehicle type. If user selects air type tanker in previous menu then only air type receivers will be shown. Moreover, if user selects land type tanker in previous menu then only land type receivers will be shown.
- **select(int vehicleID):** Selects a vehicle by its index in List, when user selects a vehicle in list by double clicking on it.
- **update(float longitude, float latitude, float altitude, float fuelAmount):** Updates the selected vehicle's related areas by calling PatikaEngine functions and show users to updated version.
- **home():** Opens Map sub-menu in MainTab main menu by calling related function from MainTab.
- **back():** Opens Tankers sub-menu in MainTab main menu by calling related function from MainTab.
- **simulate():** Opens Simulation sub-menu in MainTab main menu by calling related function from MainTab. If no vehicle is selected, this function is disabled to prevent wrong flow of program. If vehicles are selected, this function is updates selectedVehicles in PatikaEngine by calling related update function. After updating vehicles this method also calls path generation function in PatikaEngine to simulate paths.

4.2.5.5 Simulation

Simulation sub-menu contains calculated shortest path visuals and simulations of them. User can select one or more Paths to play simulation of Paths.

Method

- **home():** Opens Map sub-menu in MainTab main menu by calling related function from MainTab.
- **back():** Opens Receivers sub-menu in MainTab main menu by calling related function from MainTab.
- **select(int vehicleID):** Enables user select a simulation by clicking on a tanker vehicle. It shows selected vehicles path in simulation part.

- **deselect(int vehicleID):** Enables user deselect a simulation by clicking on a tanker vehicle.
- **play():** Plays all selected Path of tanker vehicles.
- **stop():**Stop current playing simulation.
- **pause():** Pause current playing simulation.
- **send():** This function triggers Server class to send each part of vehicles simulation to related device connected to server.

4.2.5.6 MapRenderer

Attributes

- **PatikaEngine patikaEngine:** Keeps engine of program.
- **float lookAngle:** Keeps looking angle of user to the map.
- **float x, float y, float z:** Keeps camera positions.

Method

- **setMap(File *map):** Gets map's DTED2 file path to load it. Loading part will be in PatikaEngine. This sub-menu only contains rendered object.
- **render():** Provide connection between program and OpenGL ES 2.0 to render map.
- **generateMesh():** Provide connection between program and mesh generator. Mesh generator will be calculated for each tanker vehicle.
- **setLookAngle(float newAngle):** Sets lookingAngle with new value.
- **setCameraPosition(float x, float y, float z):** Sets camera positions.
- **zoom():** Zooms the camera to the map.
- **left():** Go left in the map when user slides map to the right.
- **right():** Go left in the map when user slides map to the left.
- **up():** Go up in the map when user slides map to the down.
- **down():** Go down in the map when user slides map to the up.

4.2.6 PathFinder

4.2.6.1 PathGenerator Class

PathGenerator Class is a static class which only calculates shortest path with given mesh constrains as an Array and Vehicles list as an Array. It returns Path data Array.

Methods

- **Path[] PathGenerate(Mesh[] meshes, Vehicles[] vehicles):** This method calculates Path data with given Mesh constraints and Vehicle list. This method calls Multiple-To-Multiple-Path, Multiple-To-One-Path, One-To-One-Path functions according to meshes and vehicles.
- **Path[] Multiple-To-Multiple-Path(Mesh[] meshes, Vehicles[] tankers, Vehicles[] receivers):** Calculate Path data with given multiple tankers and related meshes to these tankers and multiple receivers. This calls Multiple-To-One-Path function.
- **Path[] Multiple-To-One-Path(Mesh[] meshes, Vehicles[] tankers, Vehicles receiver):** Calculate Path data to one receiver with given multiple tankers and related meshes to these tankers. Calculates all Path costs by using One-To-One-Path function and gets shortest path.
- **Path One-To-One-Path (Mesh meshes, Vehicles tanker, Vehicles receiver):** Calculates a Path to a receiver with a tanker and related mesh data with this tanker.

5 System Architecture

A description of the program architecture is presented here.

5.1 Architectural Design

PATİKA has six main components; GUI, Engine, Factory, Connections, Model and Pathfinder.

In our application, GUI is the main component that triggers other components respectively. First GUI screen user interacts with is the map selection tab. After this step, there is vehicle selecting and showing the generated scenario. All these operations trigger PATİKA Engine component, which in return utilizes necessary components for the selected operation.

There is one important detail in our architectural design; we are designing software for two different user profiles, one for the commanding officer and one for carrier vehicle's operator. Commanding software is considered as the main software described in this document. Operator software will be the restricted version of the main software, where users will not be able to manipulate or create scenarios. This restricted software will only be able to play the received scenarios.

The component that connects two types of users is Connections. This is the component that uses WiFi to establish a connection between the commanding software and operator software, and that enables commanding software's user to send created scenarios to the operator software's user.

More detailed information on how the components work, how they interact with each

other will be given in Detailed Design Report.

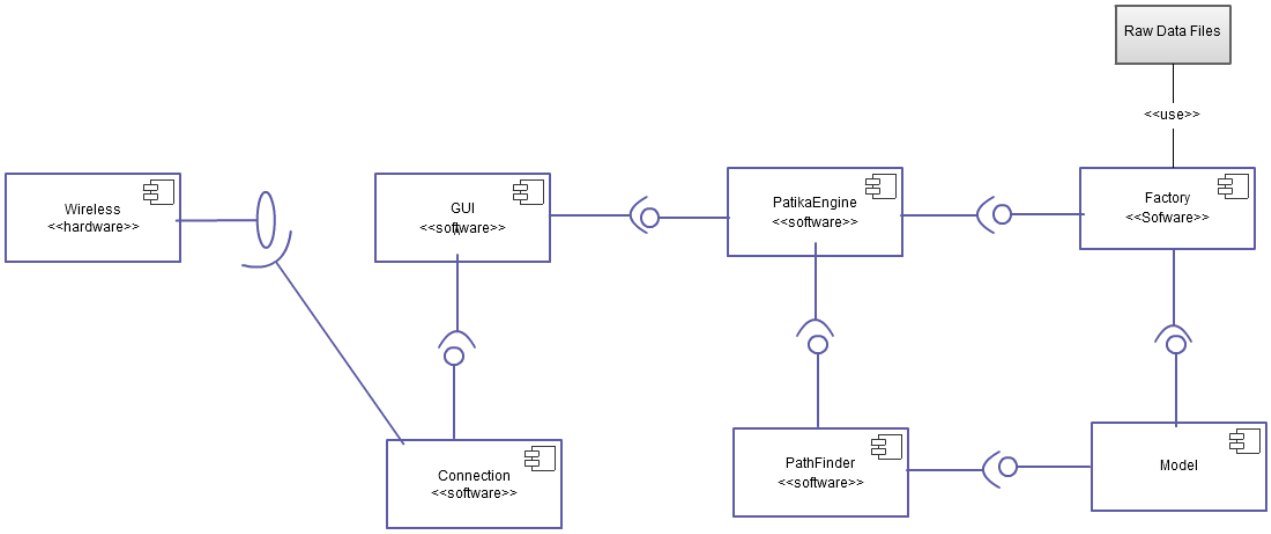


Figure 1.1 Component Diagram for Commander Type User Application

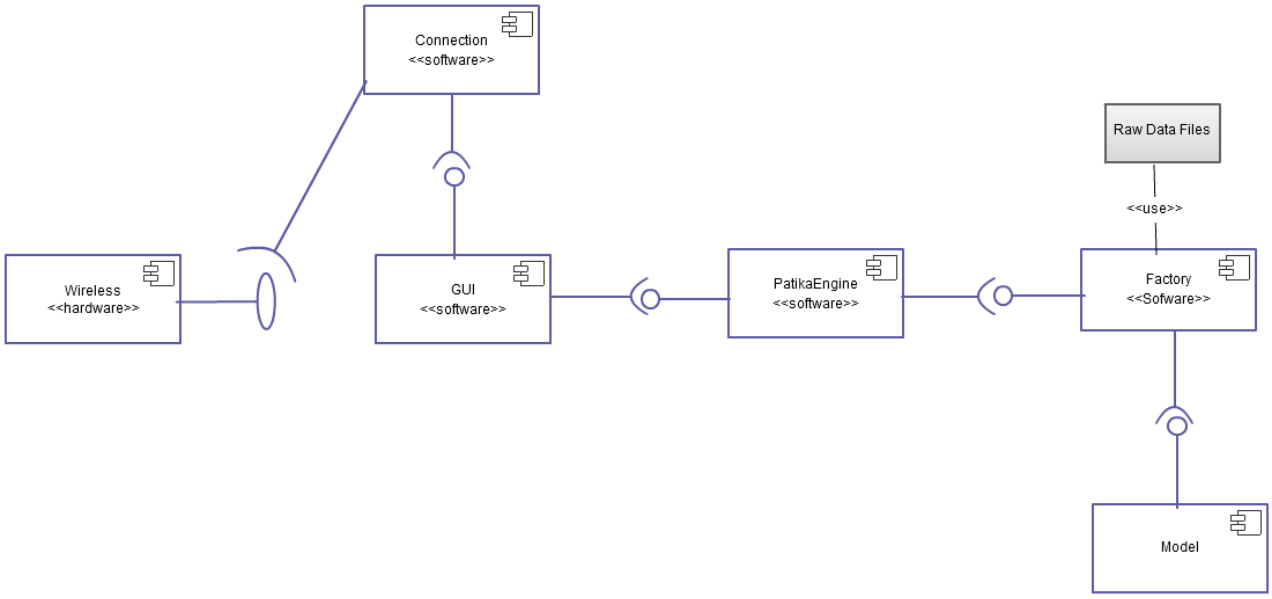


Figure 1.2 Component Diagram for Task Force Type User Application

5.2 Description of Components

The package diagram included below intends to show object-oriented nature of the basic software components of the whole PATİKA project. Implementation of the project will be done using Java; so, software packages are arranged suitably with Object-Oriented coding. All packages import the necessary classes and packages as indicated by the diagram.

Commanding user and operator user represent the two user profiles of the system. They are connected through the Connections packages of their own.

GUI package imports Engine package, which in turn imports Factory and Pathfinder packages. Factory package imports Model package. As a result, GUI is the main component that triggers a ripple effect through various components at a given time.

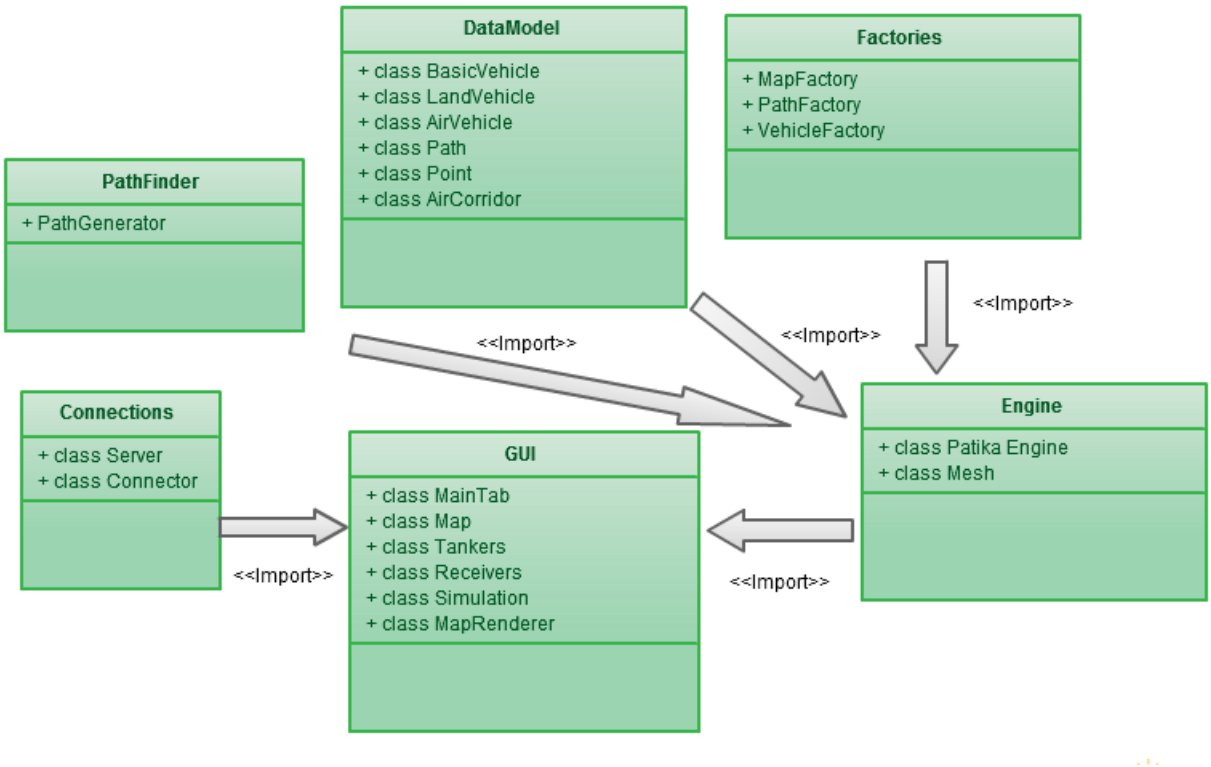


Figure 2 Subsystem Model

5.2.1 DataModel

Diagram for DataModel

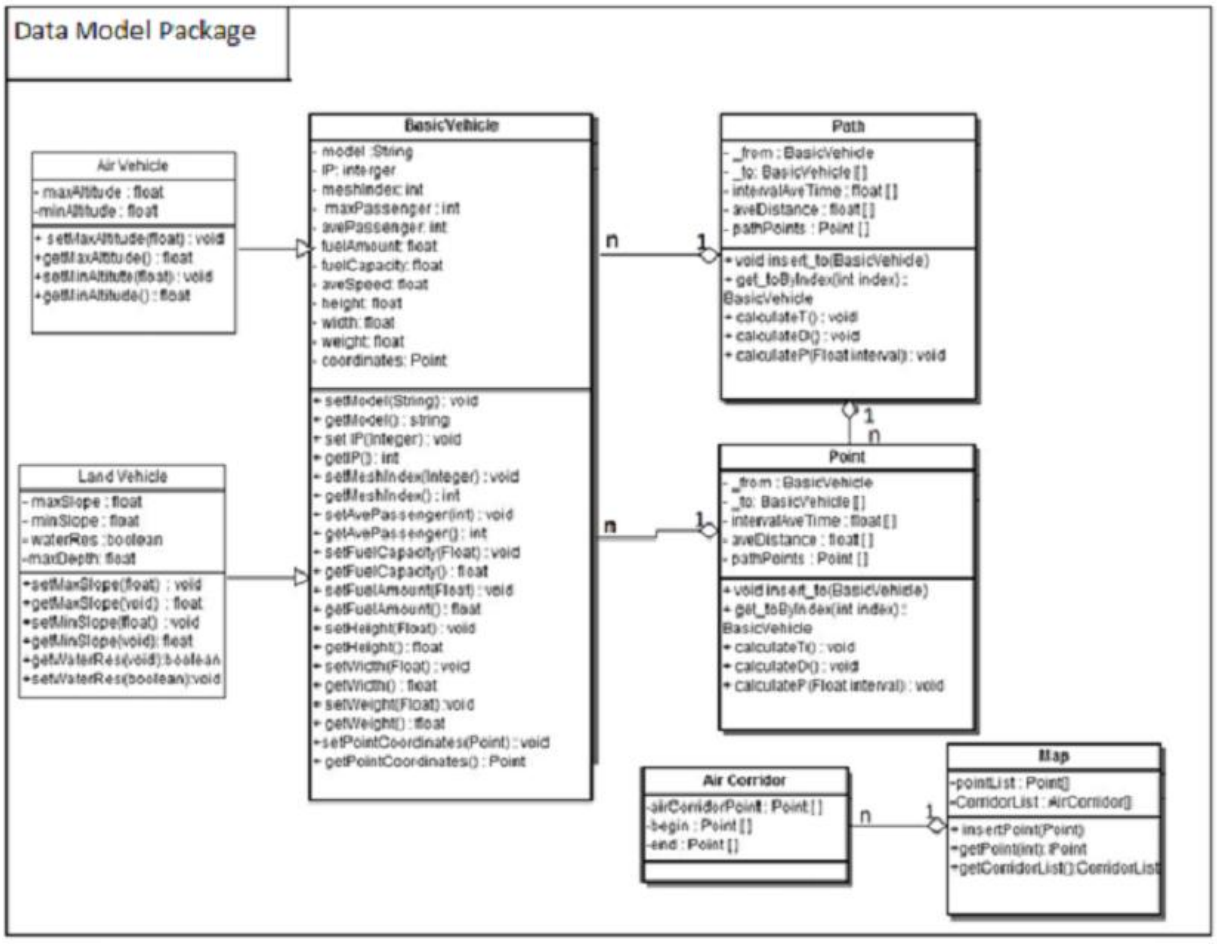


Figure 3 Class Diagram for DataModel

5.2.1.1 Processing narrative for component DataModel

This component is used for constructing main objects such as vehicle, path and points in order to achieve user interface needs. When interface class needs an objects and operations related to model package, system engine reaches the model package with the help of the factory classes and desired constructions and operations is done. The main functionalities of this component are constructing and passing vehicle, path, points and air corridor objects to the system. These objects are needed for visualization, user interface operations and path finding operations.

5.2.1.2 Interface Description for component DataModel

This component is base component for the project which contains data structures that will be used almost from any other components. However, mostly factory component interact with model component. The component has several inputs. First input comes from VehicleFactory, it reads data from file and calls all set methods in BasicVehicle class and other vehicle classes according to vehicle type. As a result of this operation an output having type BasicVehicle is generated. Second input comes from MapFactory which will read DTED2 file and give data retrieved from file to set methods of Map class. A map object is instantiated as a result of operations performed using this input. The third input type is attributes related with Path class, the PathFinder component creates Path objects and sets their attributes and this Path objects is third output for model component. In this part, for the sake of simplicity, the related input groups are described as one.

5.2.1.3 Processing Detail for component DataModel

Since there are many classes and many operations performed of these classes in this component, the processing details will be explained briefly for main classes. Factory components are used for generating meaningful instances of classes in Model. The MapFactory read map data from file, according to each point's coordinate data a Point objects is generated and Points array of Map class is created with these points. Then after creation the Map object is stored in PatikaEngine class. In addition to map, PatikaEngine needs vehicles in order to GUI can display them on the screen. PatikaEngine calls read function of VehicleFactory. VehicleFactory reads vehicle file and for each data it uses BasicVehicle class. The path is generated after PatikaEngine ask pathFinder to generate a path. PathFinder calculates the shortest path and according to the data it creates new Path. This path is stored again in PatikaEngine.

5.2.1.4 Dynamic Behavior for Component DataModel

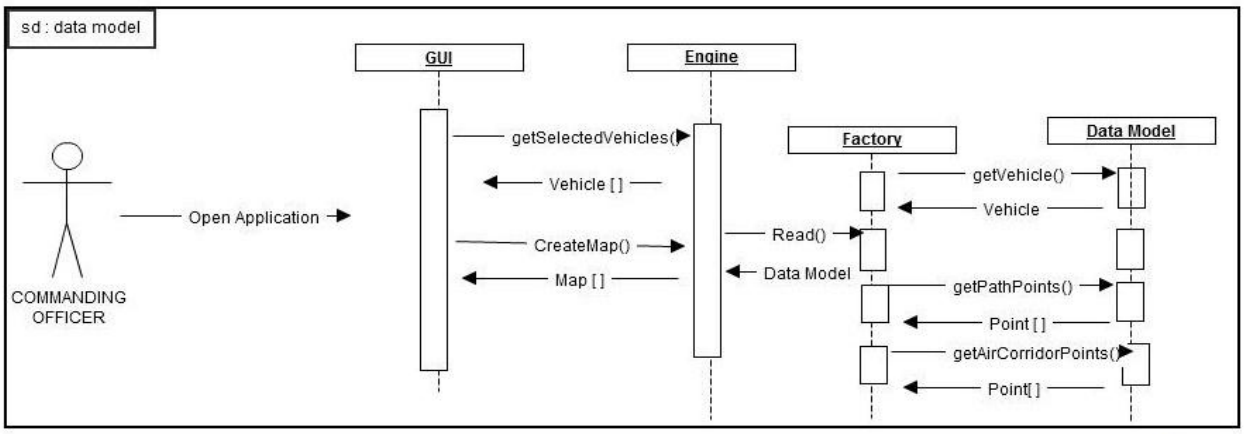


Figure 4 Sequence Diagram for Data Model (User 1)

When commanding officer user opens application, first of all he/she selects the map that will be used, on the GUI. Then, GUI package calls the CreateMap() from the engine package. In order to create map, system needs to map point data. Therefore, engine calls read() method of the MapFactory package and MapFactory calls getPathPoints() and getAirCorridorPoints() methods. After creating suitable point array list according to input, Data Model package returns point array list and gives this list to the MapFactory as an output. After that, engine generates maps and sends this map data to the GUI which displays it on the screen. Secondly, user selects the vehicle modelNo on the GUI. Then, GUI package calls the getSelectedVehicles() from the engine package. In order to load desired vehicles, system needs to vehicle objects. Therefore, engine calls read() method of the VehicleFactory package and VehicleFactory calls getVehicle() method. After creating suitable vehicle objects according to input, Data Model package returns this objects and gives them to the VehicleFactory as an output. After that, engine generate vehicle list in order to calculate shortest path algorithm and send this data to the GUI which display it on the screen in the scenario part.

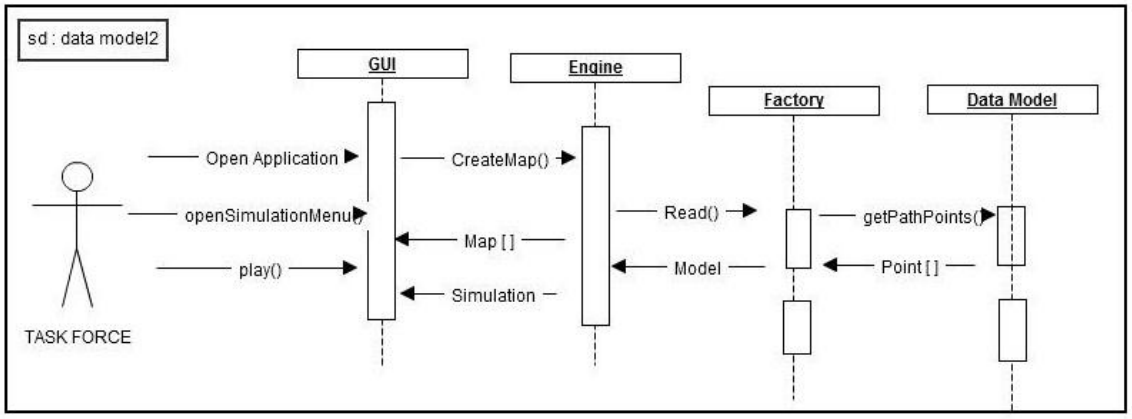


Figure 5 Sequence Diagram for Data Model (User 2)

When task force user opens application, he/she opens simulation page directly. Then, according to loaded information about the map and vehicles, GUI package calls the CreateMap() from the engine package. In order to create map, system needs to map point data. Therefore, engine calls read() method of the MapFactory package and MapFactory calls getPathPoints() method. After creating suitable point array list according to input, Data Model package returns point array list and gives this list to the MapFactory as an output. After that, engine generates maps and send this map data to the GUI which display it on the screen. Meanwhile, engine generates other path points lists, which are necessary for displaying shortest path ways on the simulation menu, by using map and vehicle information data .It finally gives simulation data to the GUI.

5.2.2 PatikaEngine

Diagram for Component Engine

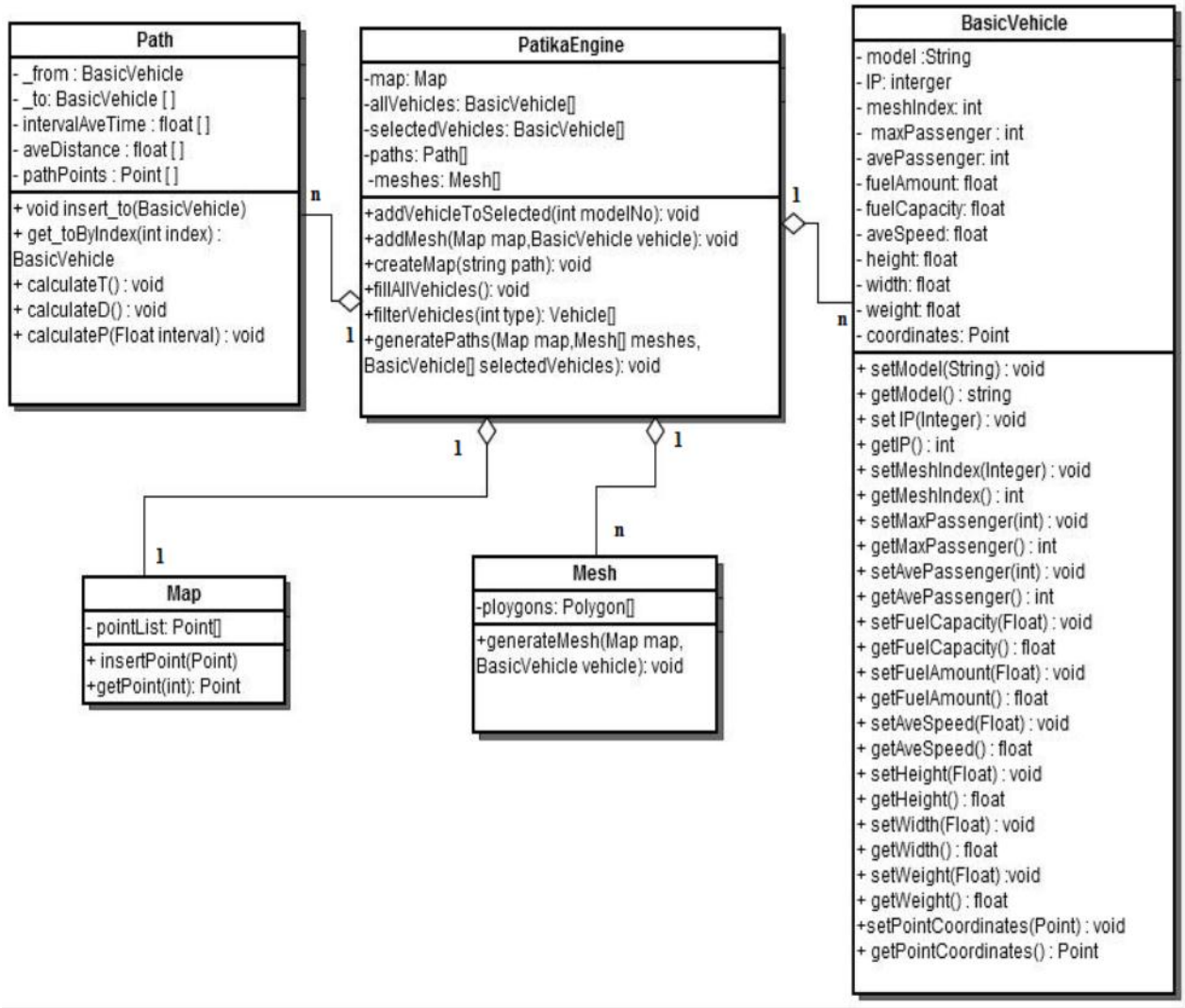


Figure 6 Component Diagram for Engine Package

5.2.2.1 Processing narrative for component Engine

This component deals with the general organization of the data that is required to visualize the scenario and the shortest path. During the stages of the application, component GUI will transfer information to and from the engine component. When users perform an action, GUI will transfer it to the engine and the corresponding functionality of the program will be triggered by it. When calculations are completed or a state is changed the engine will notify the GUI. Therefore, it can be said that the engine will be a bridge between GUI and other parts of the project. The visualization of the map and the paths will be performed by

using the Map and Paths data contained in this component.

5.2.2.2 Interface Description for component Engine

The behavior of this component depends on inputs coming from GUI. Content that will be displayed by GUI relies on the data stored in this component. The first input that engine needs is path at which DTED2 file is stored, this is necessary to generate the map information. The user will select a map from first map selection menu and the corresponding file path will be sent to engine component. The second input of engine is modelNo of selected vehicles. There is also another input, a signal which will cause engine to trigger the path finding functionality. The allVehicles attribute is output to GUI component; the GUI will display to user a list, which will contain the data stored in allVehicles. The map is also an output for GUI; it will take the map information and display it on the screen. After map is displayed and user finishes vehicle selection, the path and vehicles will be displayed on the screen and, during this process the GUI will need outputs selectedVehicles and paths.

5.2.2.3 Processing Detail for component Engine

The engine component needs the path information of the map that will be generated before performing meaningful actions. After the user selects a map in the GUI, the GUI component send path information and map is generated. Then engine sends the path and allVehicles data and GUI displays them. After each selection of vehicle performed by user, GUI notifies engine and the vehicle is added to selectedVehicles and there are displayed by GUI on the corresponding coordinates on the map. When selection is finished and the user presses finish button, the engine triggers path finding functions. The results are stored in paths attribute. Then the paths are shown on the map by GUI component.

5.2.2.4 Dynamic Behaviour for Component Engine

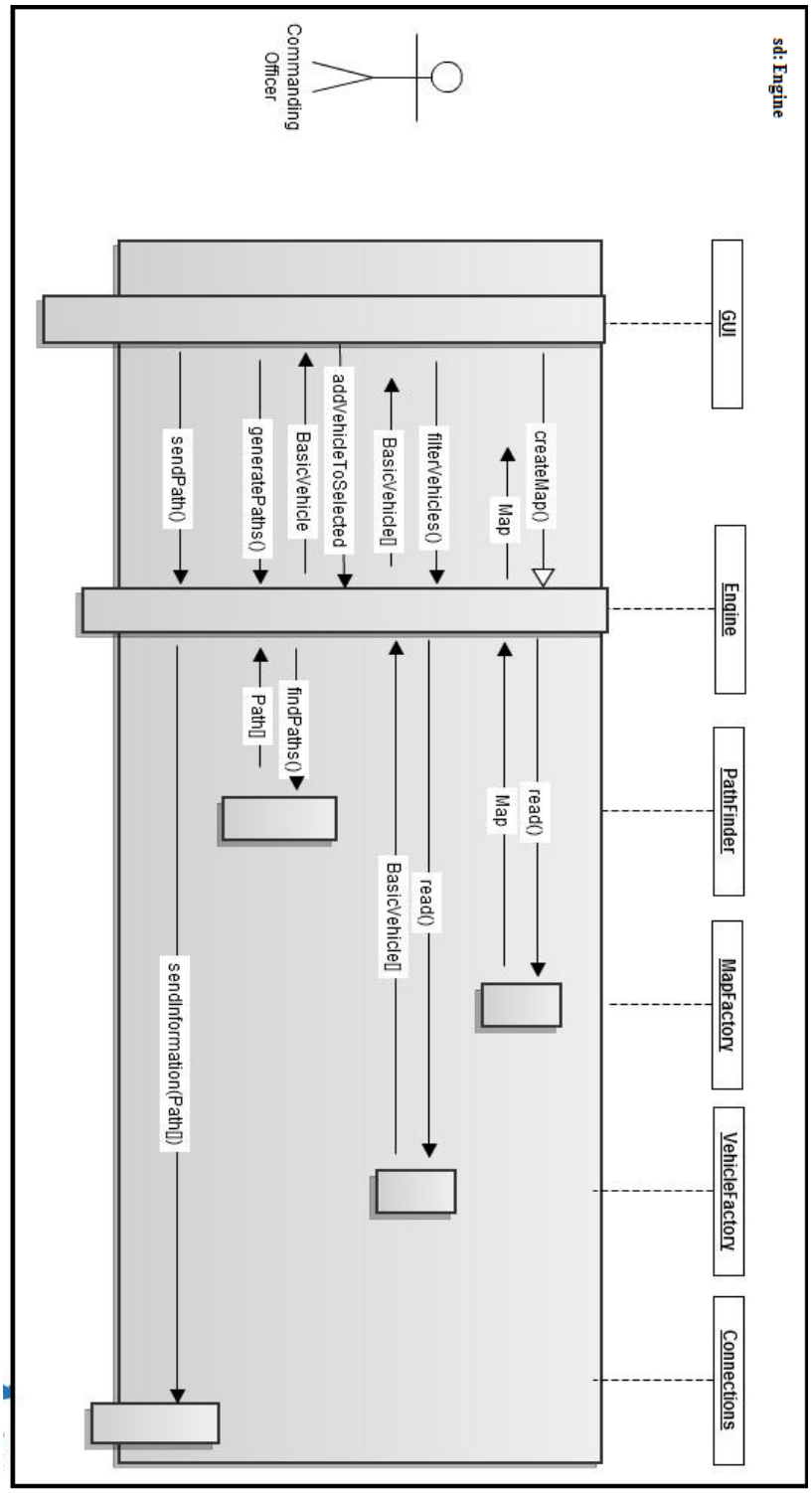


Figure 7 Sequence Diagram for Engine Package (User 1)

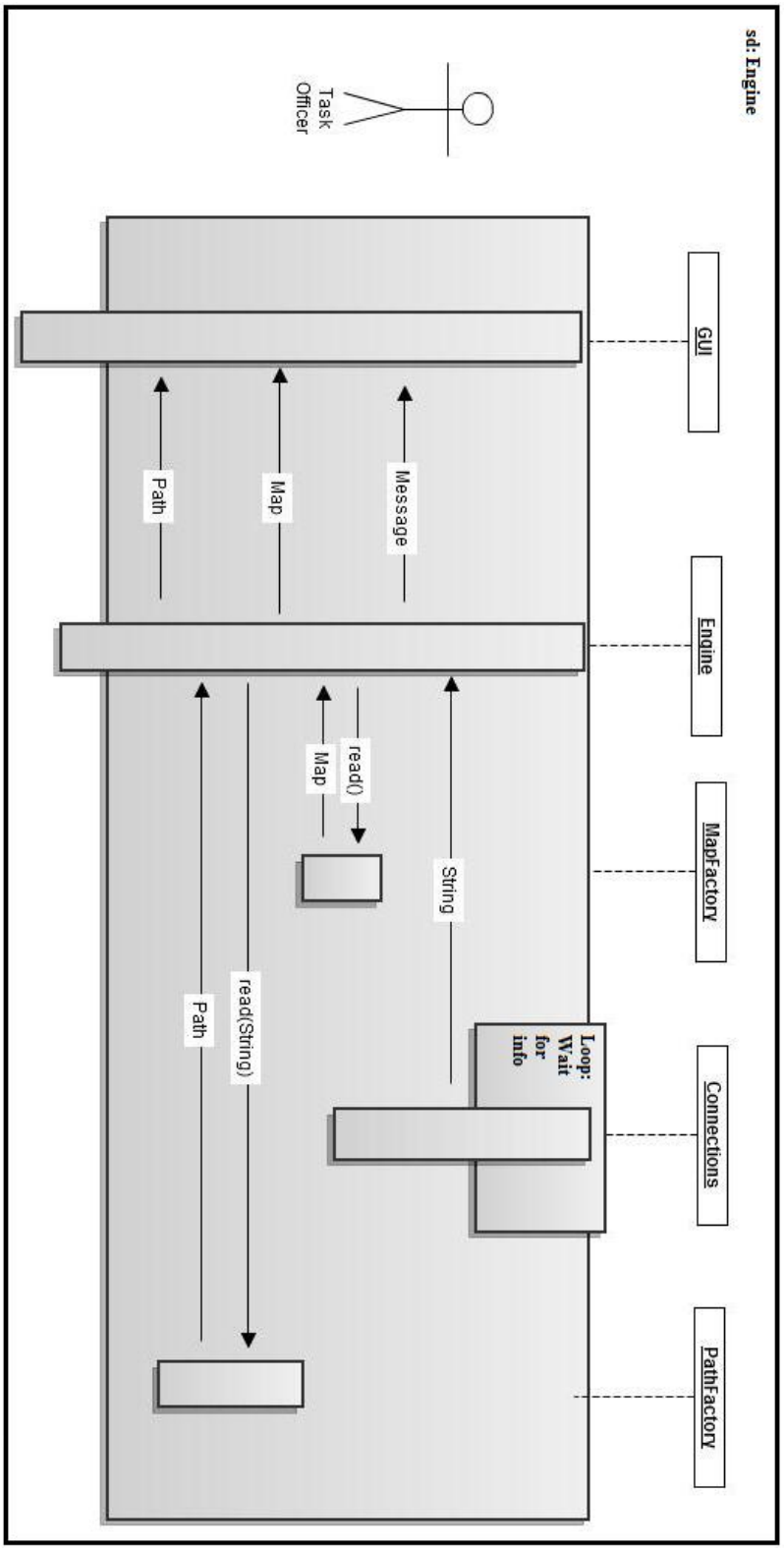


Figure 8 Sequence Diagram for Engine Package (User 2)

When commander officer Ali opens the application, he will be presented with a screen which will contain possible maps that he can select. Then after he tabs the screen, the GUI package will inform Engine package by calling createMap function. In this function read function of MapFactory class in Factories package will be called and a Map object will be created. The object will return to Engine and then, it will returned to GUI in order to MapRenderer can render the map. Then Ali will be in the second stage at which, he can see the map he was selected in big screen and he could be able to zoom in/out and travel across to map. In this screen Ali will see a menu on the left in which vehicles of tanker type will be shown. However, before these menu is shown GUI will call filterVehicles function and that function call read function of VehicleFactory class of Factories package and it will will return the vehicles. Then the filterVehicles method will filter them according to whether they are tanker or not and will return vehicles that are tankers. Then, Ali will able to see tanker a the left side of the screen. The commanding officer Ali will select the tanker vehicles, enter their coordinates and then press the next button. During the selecting vehicle stage, addVehicleToSelected function of Engine will be called and selected vehicles will simply added to some other array. The corresponding array will returned to GUI, so Ali can see the results of his selections. Then, Ali will press the next button and a screen for selecting receiver vehicles will be opened. In this stage again filterVehicles and addVehicleToSelected function will be called. Then, Ali will want to see shorthest path between the vehicles and when he presses the next button, generatePaths function of Engine will be invoked. This function will call findPaths function in PathFinder and findPaths will return paths array. This array will also be transferred to the GUI in order to be rendered on the map. When commanding officer Ali, wants to send path information to the task officer, he will press the send button. Then, sendPath function of Engine will be called and it will send information by calling sendInformation function of Connections class.

By the way, the task officer Mehmet will be waiting for path information that will come from his commander. Then, when information comes he will see a message on the screen which is received by Connections class and transferred to Engine and then to GUI. Then, the Engine will call MapFactory using information come from commander's application. The MapFactory will create a Map and pass it to the Engine, then it will send to GUI and MapRenderer will render the map. Then, PathFactory is called by Engine with string that come from other user it will generate a path and send it to the Engine and Engine will send it to GUI. Finnaly, Mehmet will able to see the sorthest path and he can use it in his task.

5.2.3 Factories

Diagram for Component Factories

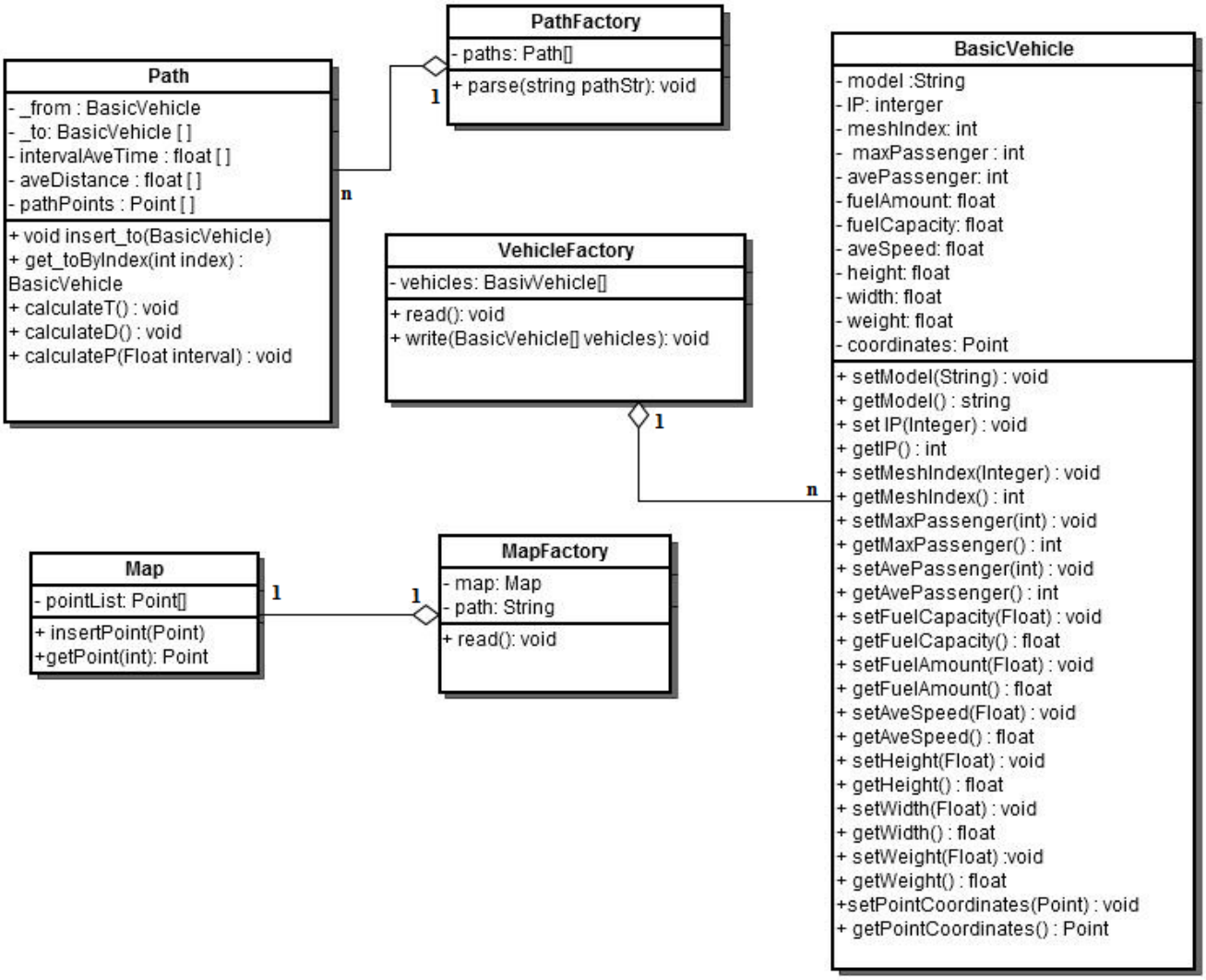


Figure 8 Component Diagram for Factories Package

5.2.3.1 Processing narrative for component Factories

This component's main purpose is to generate instances of specific classes which will be used by the Engine component. The corresponding classes are Path, Map and BasicVehicle. The Map and BasicVehicle instances will be generated by reading the information from file, while the Path class will be generated by parsing a string. There will be a write option for BasicVehicle instances, this option will be used when user needs to change features of the vehicles.

5.2.3.2 Interface Description for component Factories

Factories component will need three inputs. First one is the file path that is needed while reading the map file. The second one is a string variable that contains path information that will be used when parsing and generating the Path [] instance. Third input, which is an array of BasicVehicle, is needed by VehicleFactory to write changed features of array to file.

5.2.3.3 Processing Detail for component Factories

The classes of factories components will be triggered in different times in order to generate the needed instances of classes in Model component. The MapFactory will be used when user selects a map from GUI. GUI will notify the patikaEngine component and patikaEngine will send file path to MapFactory. After map generation is completed the Map attribute in patikaEngine will be assigned to Map generated by MapFactory. PathFactory will be used when application receives new path information from another application. The incoming data which will be in string formats will be parsed by PathFactory and new Path variables will be instantiated. VehicleFactory will be used when GUI component needs to display available vehicles. GUI component will request vehicles from patikaEngine and patikaEngine will call VehicleFactory.

5.2.4 Connection

Diagram for Component Connection

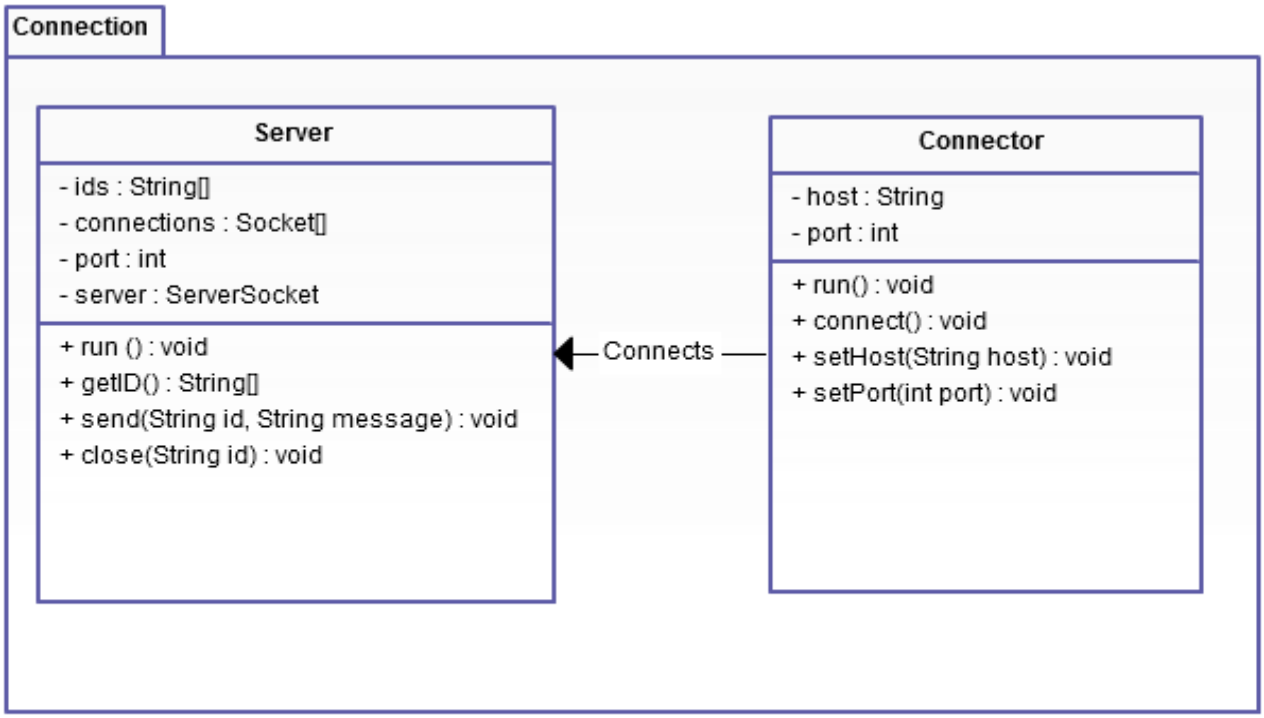


Figure 9 Component Diagram for Connections Package

5.2.4.1 Processing Narrative for Connection

Connection component handles connection between Command Device and Mission Device. Assume having secure connection over Wi-Fi is enabled Connection connects two devices and manages send and receive messages. Connection processes as a background task. Server class handles waiting and accepting connections. It is also responsible for sending message to all connected devices or specific ones. Connector class manages connecting a defined server and waiting – accepting message from that server.

5.2.4.2 Connection Interface Description

The input interface for Server sub-component is GUI that gets actions directly from user to send calculated Path to connected devices. The output interface is also GUI that shows user to percentage of sending message progress. The output interface is also Mission Device's Connector sub-component by sending message to it.

The input interface for Connector sub-component is Command Device that send message to this component. The output interface is GUI that shows received calculated Path data.

5.2.4.3 Connection Processing Detail

When Command Device program is running:

- Server class starts Socket server as a background program and waits for connection.
- Accepts connections and keep the current list of all connected devices.
- Waits a send command from user via GUI
- Sends messages to the connected devices

When Mission Device program is running:

- Connect the specified device.
- Waits a message from connected device.
- When message is received, it triggers GUI with received Path data to show user simulation.

5.2.4.4 Dynamic Behavior Component Connection

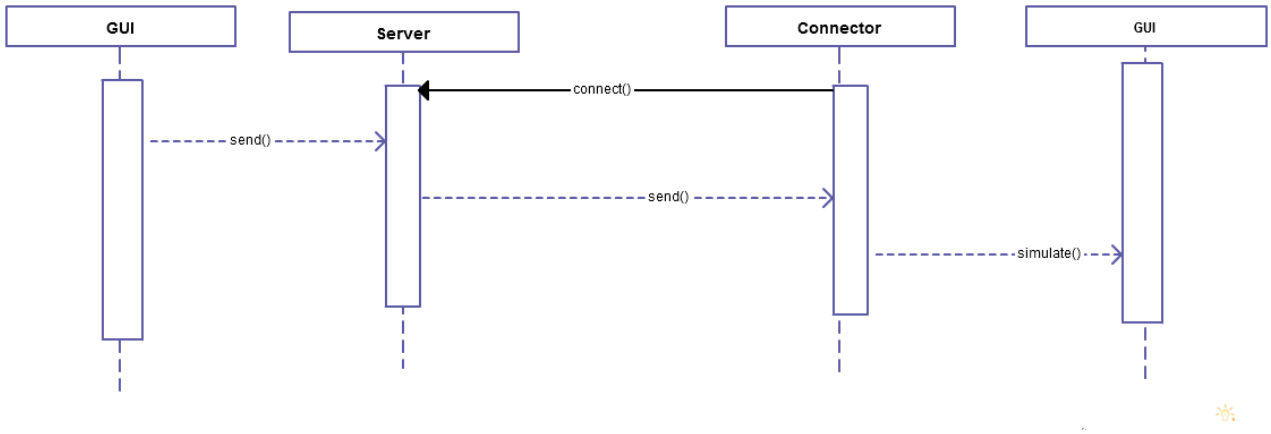


Figure 10 Sequence Diagram for Connection Package

Commander Officer Ali opens program and in background Server class in Connection package accepts connection that coming from any Task Force user. In Task Force user side, when Task Force user opens application, Connector class in Connection package connects specific first type user in this case Commander Officer Ali.

When the Commander Officer opens program, simulates path and presses the Send button to send Task Force type users, Server class in Connector package sends the calculated Path data to the connected users. If there is multiple path data for multiple users, it sends to data correct user. In Task Force side, when Path data is received, Connector class triggers GUI and simulated Path data

will be shown to show Task Force users.

5.2.5 GUI

Diagram for Component GUI

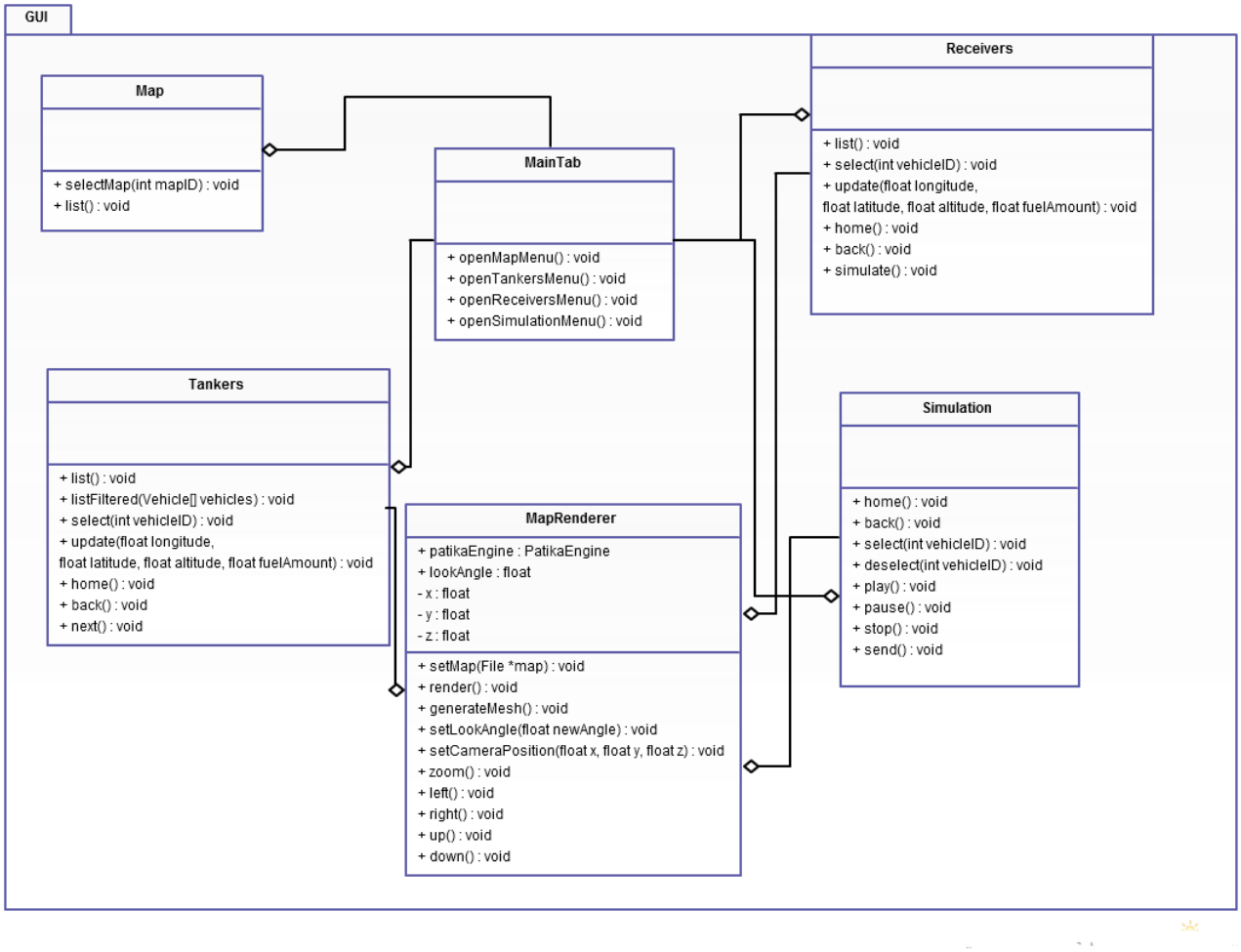


Figure 11 Component Diagram for GUI Package

5.2.5.1 Processing Narrative for GUI

GUI is responsible for showing data as visualized to user and get actions from user. It also controls flowing of programming by enabling reachable parts and disabling unreachable parts of program. Enabling transitions between tabs and showing current state of program is provided. This package gets actions from user and starts related engine function or server functions to work program correctly.

5.2.5.2 GUI Interface Description

The input is mainly user interactions for GUI package because it manages interaction between user and program. It waits users' events on program. Moreover, PatikaEngine is the second main input interface for GUI which activated by related updates functions. Connector class is also input for GUI by receiving message from Command Device and simulating it on GUI.

The output is mainly user for GUI package because it shows and simulates data to users. Moreover, Moreover, PatikaEngine is the second main output interface for GUI by activating related update functions. Server class is also output for GUI by sending messages to the connected devices from Command Device.

5.2.5.3 GUI Processing Detail

GUI shows menus to the user and visualizes data to user to be more attractive and user-friendly. It lists all vehicles in two sub-menus by splitting them into two lists by filtering vehicle type. It also renders the 3D map that data come from DTED2 file to. It also simulates Path data and enables send option to user. This feature is used for divide to parts of Path data. It is also used by sending separate and related parts of Path to related connected device. This splitting process is transferred to user correctly and understandably by GUI.

GUI gets actions from user by listening interaction of user. According to type of action GUI calls PatikaEngine class functions so update operations and flow will be done.

5.2.5.4 Dynamic Behavior of GUI

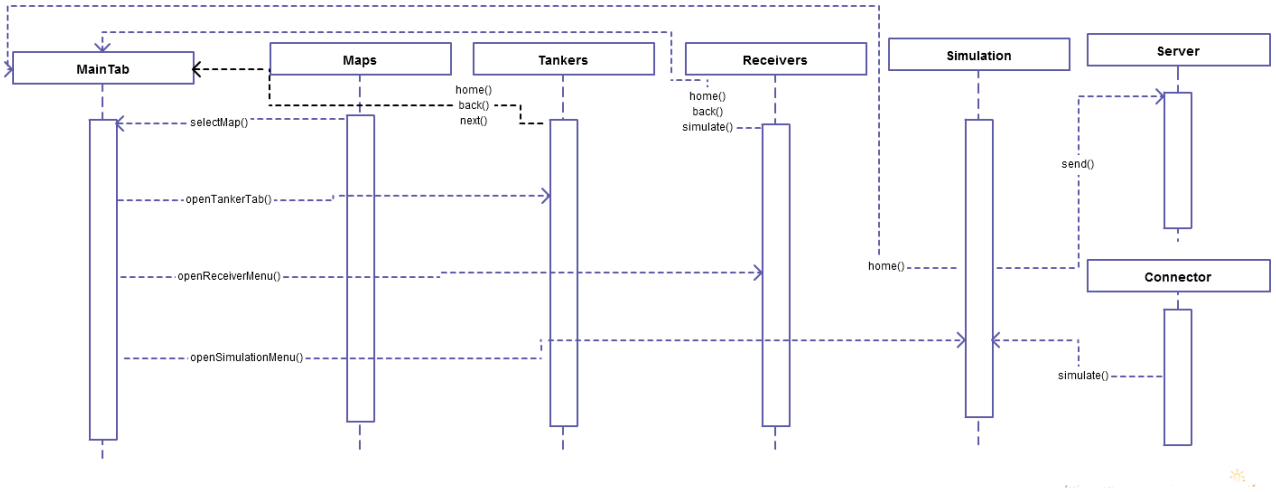


Figure 12 Sequence Diagram for GUI Package

When Commander Officer Ali opens application, he sees Main Tab menu which contains Tab Buttons to reach sub-menus which are Maps, Tankers, Receivers and Simulation Menus. At the beginning Commander Officer Ali will see grid list of maps in the system. In this sub-menu if Commander Officer Ali clicks any Tab Button to change menu, action will not get any response. However, Commander Officer Ali double - clicks any map object on Maps sub-menu screen and after some waiting time Tankers sub – menu will be opened. In Tankers sub – menu, Commander Officer Ali will see list of Tankers in the system and rendered 3D map in addition to Tab Menu buttons. Commander Officer Ali will only go back when he presses Maps Tab Menu button, otherwise he will not get any response. When he presses Home button, program goes back to the Maps sub – menu. After selecting Tankers and then pressing Next button, program opens Receivers sub - menu. In Receivers sub – menu, Commander Officer Ali will see list of Receivers in the system and rendered 3D map in addition to Tab Menu buttons. Commander Officer Ali will only go back when he presses Maps Tab Menu button or Tankers Tab Menu button; otherwise he will not get any response. When he presses Home button, program goes back to the Maps sub – menu. After selecting Receivers and then pressing Simulate button, program opens Simulation sub – menu after some waiting time. In Simulation sub – menu, pushed any Tab Menu button will give a response by opening clicked sub – menu. In Simulation sub – menu, Commander Officer Ali can play, pause and stop simulated paths. When he clicks Send button, Path data will send to connected Task Force users via Connector package.

When Task Force user Mehmet opens program he will see only waiting screen before receiving Path data. After receiving Path data he will see Simulation sub – menu content with simulated path data on rendered map. Task Force Mehmet will also be able to play, pause and stop simulation.

5.2.6 PathFinder

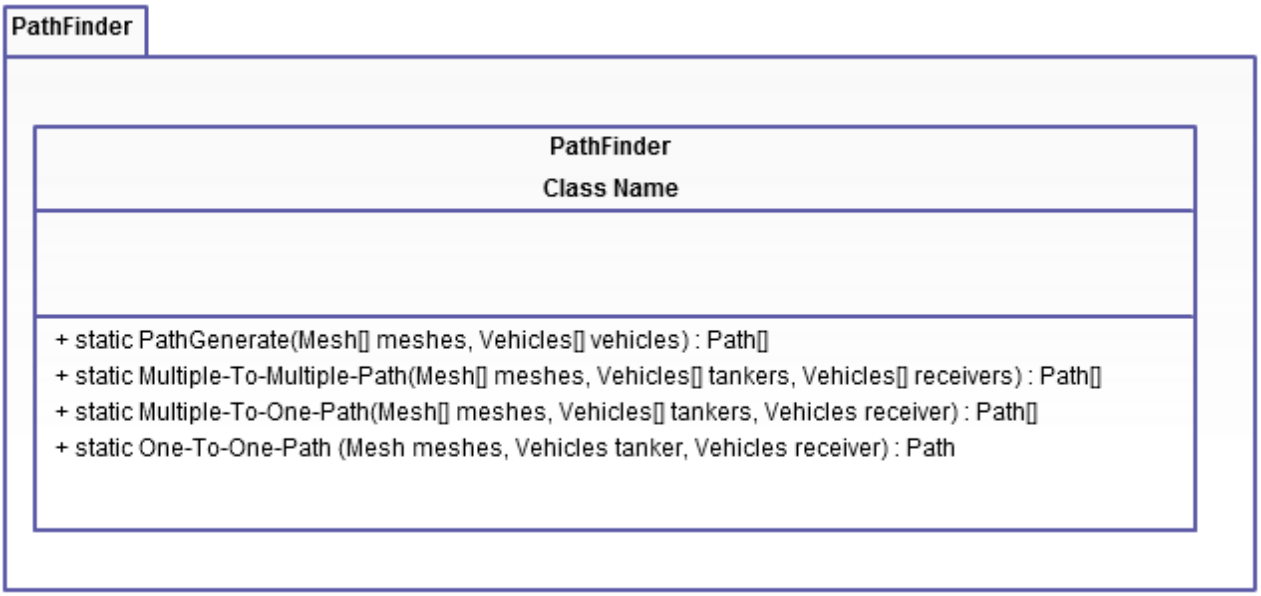


Figure 12 Component Diagram for PathFinder Package

5.2.6.1 Processing Narrative for PathFinder

PathFinder is shortest path calculator with given mesh constraints and vehicle list. Functionality is split to its parts according to vehicle count. Shortest path algorithm will be implemented in this component.

5.2.6.2 PathFinder Interface Description

The input interface is PatikaEngine which start simulation. When user makes an action to get simulation of path in GUI, generation of simulation will be called explicitly. The output interface is also PatikaEngine which waits result of simulation after called generation. After calculate shortest path PathFinder returns Path data to PatikaEngine.

5.2.6.3 PathFinder Processing Detail

- User demands simulation of path by clicking a button. Then PatikaEngine is triggered to call PathGenerate().
- PathGenerate will call necessary functions according to vehicle count and then collect results from each function to create overall Path data.
- Generated Path data is returned to PatikaEngine to be shown in GUI.

5.3 Design Rationale

As we stated above, our design is composed of six main components, namely GUI, Engine, Factory, DataModel, PathFinder and Connections. Since we keep the information about data objects in raw data files (xml), we needed a component for reading these files and filling DataModel classes with the data. We named it Factory. Our data objects, whose detailed information will be kept in DataModel package and its classes. They will be re-initialized by Factory component on every start of the application.

Our system will do data operations without needing user prompt, but after these operations are over, system will wait for user input. We named this main triggering component GUI since we will always communicate with the user through graphical user interface. GUI has six sub components which are responsible for handling the different DataModels such as vehicles, air corridors, maps...

To encapsulate all GUI triggered operations, we created a component named Engine. Engine will be responsible for handling all requests made by GUI, by activating the needed components such as Factory, PathFinder...

Finally, we created a Connections component to handle Wifi usage of the users.

While dividing the components into sub components, we tried to divide the system into meaningful parts; however, we tried not to divide it unnecessarily. We took into consideration that each sub component shares some features, but also they have different goals than each other.

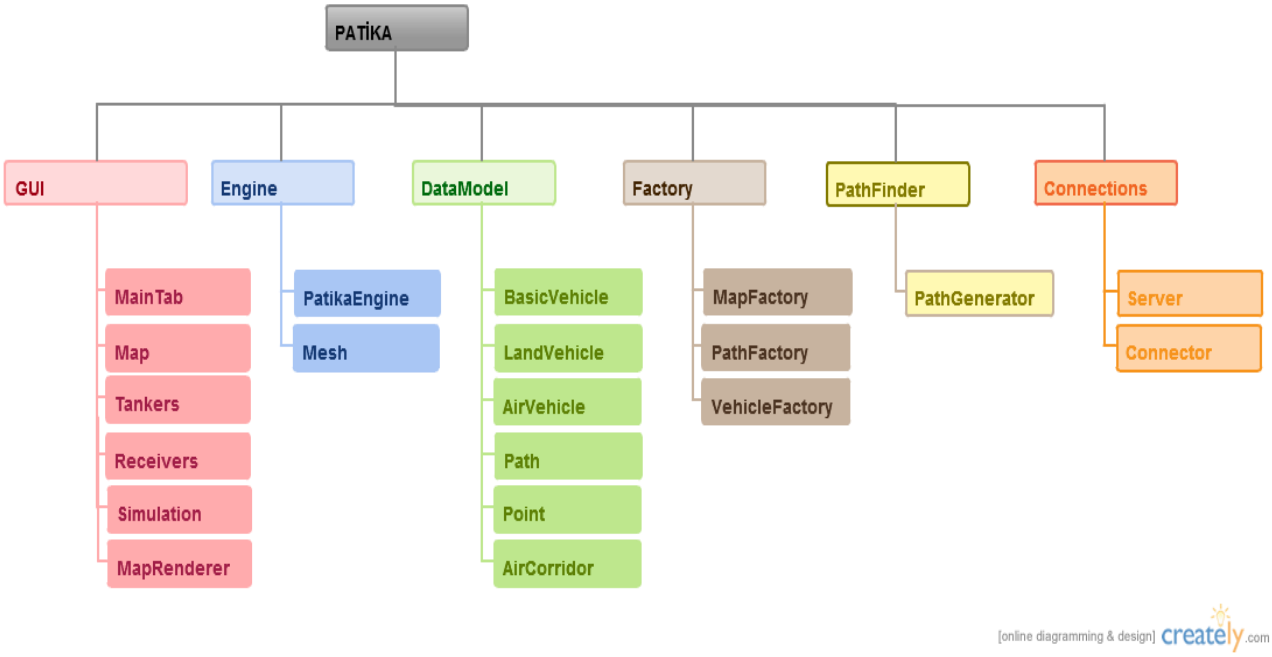


Figure 13 Aggregation Hierarchy Diagram for PATİKA

5.4 Traceability Matrix

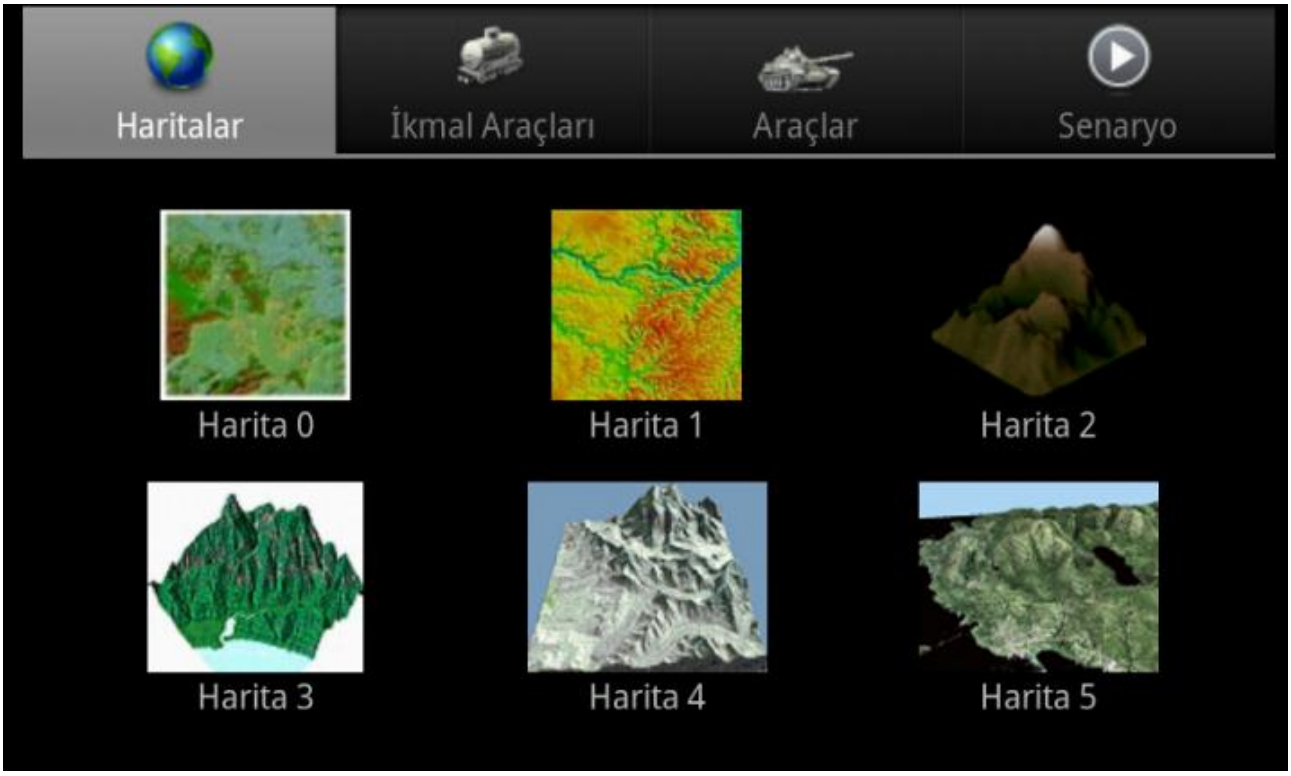
SRS	DDR
3.2.1	5.2.1, 5.2.2, 5.2.3, 5.2.5
3.2.3	5.2.5, 5.2.6, 5.2.2
3.2.4	5.2.5,5.2.1
3.2.5	5.2.5, 5.2.6, 5.2.2, 5.2.1, 5.2.3

After the SRS we conducted a meeting with ASELSAN and made extensive changes on our design of the project; thus, an accurate traceability matrix is impossible to implement at this point. We tried to match the cases as well as we could.

6. User Interface Design

6.1 Screenshots

6.1.1 Map Selection Screen



On this screen, all available maps are shown to user. User will select the desired map on which to generate a scenario. When selected, user will be transferred to the next screen automatically.

6.1.2 Carrier Vehicle Selection Screen

In this screen, user will choose the desired fuel carrier vehicles from the side bar. When a vehicle is chosen, users will be able to enter their coordinates either manually as will be shown in the next figure or by dragging the vehicle from the menu to the desired location on the map and then dropping it. Without selecting any vehicles from this menu, the next button will not be activated. When user completes the fuel carrier selection part, he/she will press the next button. At any given time, user may press the back button.

6.1.3 Reciever Vehicle Selection Menu

It will be the same as the fuel carrier selection menu, but instead of carrier vehicles, reciever vehicles will be shown in the side menu.

6.1.4 Simulation Menu

In this menu, there will be a full visual of the map and the added vehicles on it. Moreover, the generated paths will be shown. At the bottom of the screen , there will be a play scenario menu, where users can play, pause or stop the simulation. There will also be a send button that sends the satisfactory scenarios to the operator devices.

7. Detailed Design

7.1 Data Model

***Classification*:**

Data Model component is a base package of the overall system .It constructs main objects such as vehicle, path and points in order to achieve user interface needs.

Definition:

The main tasks of this component are constructing and passing vehicle, path, points and air corridor data objects to the system. These objects are needed for visualization, user interface operations and path finding operations. It takes inputs about the desired data object from the factory package and gives necessary output to it again.

Responsibilities:

The component is responsible from generating data structures that will necessary for almost all other components. By providing the map and vehicle information which is needed to display the map and simulation part, its plays a constructer role in the system such that it constructs point, path and vehicle class and gives to the system.

Composition:

This component consists of five classes, BasicVehicle, AirVehicle, LandVehicle, Point and AirCorridor. BasicVehicle class is keeping common information about the vehicle and is used for generating AirVehicle and LandVehicle objects. By using fuel amount, fuel capacity and coordinate fields, it affects the path find algorithm in different ways. AirVehicle

and LandVehicle classes are extended from the BasicVehicle class and specialized for supplier/carrier vehicles which are designed to move on the air or land. In the LandVehicle object slope and water resistance information is kept and according to these data the path, that supplier vehicle can follow, is found. In the AirVehicle object altitude information is kept and according to these data the path, that supplier air vehicle can follow, is found. Point class is keeping common fields, such as height, latitude, longitude and normal vector. It is necessary for generating map and shortest path list data. AirCorridor class is keeping information about the some key points. By using this, path generator package determine the path of the air vehicle supplier.

Uses/Interactions:

This component has interaction with mostly Factory component of the system. It takes several inputs from the Factory and gives desired outputs to it. First of all, vehicle object is generated according to request from the VehicleFactory class. It reads data from file and calls all set methods in BasicVehicle class and other vehicle classes according to vehicle type. Then, it takes vehicle object from the Data Model package in order to accomplish several tasks. Secondly, point objects are generated according to request from the PathFactory class and MapFactory class. PathFactory needs path list which will be used in path generator class from the data model package and these points are constructed and supplied back to PathFactory by data model package. MapFactory will read DTED2 file and give data retrieved from file to set methods of Map class. A map object is instantiated as a result of operations performed using this input. Data Model package returns point array list and gives this list to the MapFactory as an output. After that, engine generates maps and sends this map data to the GUI which displays it on the screen.

Resources:

The resources used by this component are physical memory and CPU. Physical memory is used to load package objects in the mobile device. CPU is used to centralize all the functions necessary for a system to perform. It retrieves information from the computer memory and executes instructions.

Processing:

When user starts the application, Factory package reads the required data from the raw data files and initializes this component. This component will not do any state changes or initializations by itself. It will always be handled by the Factories package. We do not expect many exceptional conditions since initialization of this component depends on xml reading and xml modification at the core. If any exception during this process occurs, Factory package will attempt a re-read of the xml file. If a read is entirely impossible, this

component's relative class will be left unfilled.

Interface/Exports:

Since this component is at the bottom of the event trigger pyramid, it will not use any components. It's for storage only.

7.2 Engine

Classification

Engine is package that will be used as a bridge between GUI and other parts of the project.

Definition

The main task of this package is to organize the data that will be used to visualize the map, vehicles, paths and scenarios. The related data will be stored and Engine package and processed by responsible packages and classes that will imported by it. Moreover, according to the signals that will come from GUI, data stored in Engine package will displayed on the screen to the user.

Responsibilities

The package has huge responsibilities because it is playing the bridge role. It should response to the signals coming from the user that will be directed to Engine by GUI. According to the type of signals, Engine will generate meaningful data that will represent map, vehicles and paths by using the other packages that are responsible from generating data. Moreover, this package will invoke the PathFinder package and the result of the shortest path generated by that package will also be stored in Engine package.

Constraints

In order to generate meaningful data, it is assumed that user will select map before demands the shortest path calculation. Moreover, it is also assumed that user will provide correct coordinates for selected vehicles such that coordinates are not outside of selected map. In addition, since Engine will provide vehicle and map information to GUI package, we also assume that corresponding data files are stored in the system and are correct, in order to Factories package to read and generate vehicles and map.

Composition

The Engine package consists of two classes PatikaEngine class and Mesh class. PatikaEngine holds the vehicles, map and paths and provides them to the GUI package. Mesh class will generate a mesh on the selected map in which selected vehicle is able to move. The mesh will be used in shortest path calculations.

Uses/Interactions

The package interacts with almost every other package in the project. It interacts with GUI to get information of which map and vehicles are selected. It also sends the calculated shortest path information to the GUI. In addition, the package interacts with Factory package, which produce data representations of vehicles and map from data files. The results produced by Factory package are stored in Engine package. The package interacts with PathFinder package, it sends the map and vehicle type information to the PathFinder and gets the path data that is calculated. The last interaction of Engine package is with Connections package in order to send and receive path information to and from other types of users.

Resources

The resources used by this package are memory and CPU. Memory is used for storing the local variables of package and CPU is used for calculations made in Package.

Processing

The GUI sends the selected map information to the Engine package. Then Engine calls Factory package and after Factory reads and generates map it sends the map to the Engine. Then, Engine sends map information to the GUI package. After that, user will selected the vehicles and Engine will be signaled. Engine will demand from Factory package, the vehicles and Factory will read data files and produce vehicles and send them to Engine. Engine will send the results to the GUI and they will displayed by GUI. Then, the user will demand to see shortest path and signal Engine and Engine will signal the PathFinder package, it will calculate the path and the result will be stored in Engine and Engine will transfer the date to GUI. At last when user want to send generated path to second type of user, again Engine will be notified and Engine by using Connections class will send the path data to other user.

Interface/Export

The package needs to use some other packages defined in the project such as Factory, Connections, PathFinder and Data Models packages.

7.3 Factories

Classification

Factories are a package that will read the date files and produce representation of it in the project.

Definition

The main task of this package is to read data files that will hold information about vehicles, maps and paths. Every class will read corresponding file and store the information as the instance of corresponding class in Data Model package. Some classes will also write information to the file when it is necessary.

Responsibilities

The responsibilities of this package are basically read from and write to data files. When Engine demands the map and vehicle to be generated, the corresponding class of the package will be called and it will read from data file, then the corresponding Data Model object will be instantiated. When user demands to change some characteristics of vehicles, again Factories will be awakened and the changes will be written in corresponding data file by this package.

Constraints

It is assumed that the data files will be stored at correct location of the disk and the format of them will correct, in order to Factories produce correct Data Model instances.

Composition

The factories package consists of three classes which are MapFactory class, PathFactory class and VehicleFactory class. The MapFactory will be used to read DTED file, create and instantiate of the Map class (Data Model package). The VehicleFactory will read the data file in which properties of vehicles will be stored and produce BasicVehicle (Data Model package) objects. I will also write to the vehicles data file when it is necessary. PathFactory will produce a Path (Data Model package) object from a data file or from a string sequence. It will be used for second type of user, to whom path information will be sent via Internet.

Uses/Interactions

The package is used by Engine for reading and writing information from data files. The package uses Data Model package, because it simply create instances of classes in that package. Factories package interacts only with Engine and Data Model package.

Resources

The data files are needed by this package in order to Data Model objects can be produced. We selected XML as the data format for Vehicle and Path, because it is fast and easy to write and read by Java. In addition, we select DTED2 as data format of the map, because it contains the coordinate information as well as height information. The example format of Vehicle file is as follows.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Vehicles>
        <Vehicle type="Land" tanker="false" name="Kamyon" IP="192.168.1.1" >
        <maxSlope>45</maxSlope>
        <minSlope>0</minSlope>
        <waterRes>false</waterRes>
        <maxDepth>0</maxDepth>
        <fuelCapacity>11000</fuelCapacity>
        <fuelAmount>1000</fuelAmount>
        <avgSpeed>70</avgSpeed>
        <width>2</width>
        <height>3</height>
        <maxPessenger>3</maxPessenger>
        </Vehicle>

        <Vehicle type="Air" tanker="true" name="Yakıt İkmal Uçağı" IP="192.168.1.3" >
        <maxAltitude>5000</maxAltitude>
        <minAltitude>1000<minAltitude>
        <fuelCapacity>10</fuelCapacity>
        <fuelAmount>50000000</fuelAmount>
        <avgSpeed>600</avgSpeed>
        <width>5</width>
        <height>15</height>
        <maxPessenger>3</maxPessenger>
        </Vehicle>
</Vehicles>
```

Processing

The Engine package calls Factories package when it needs Map object. MapFactory class read the corresponding DTED file created the objects and sends it to Engine package. When engine package needs the vehicle list, again it calls the Factories package and VehicleFactory creates vehicle list by reading vehicles data file. When user needs to change properties of a vehicle, Engine is notified and it calls Factories, then VehicleFactory writes changes to the vehicle data file. In addition, when first type of user sends the path information to the second type of user, Engine will again call the Factories and PathFactory will read the string that come from other user and create Path object.

Interface/Export

The package needs Data Model package and data files stored in order to produce results.

7.4 Connection

Classification

Connection is a package in this system. This package manages connections between defined two type users.

Definition

Communication between two type users is stated before. So connection package contains functionalities for that purpose. Commanding Officer type user can send calculated path message to the Task Force type user by functionalities that are given by Connection.

Responsibilities

Server class of this package is responsible from sending message to the Task Force type user. Arrangement of message and receiver is another responsibility of Server class of Connection.

Connector class of this package is responsible from receiving message from Commanding Officer type user. Triggering GUI package to simulate calculated path that are received by message is another responsibility of Connector class of Connection.

Constraints

WiFi connection is an assumption for this package. It is assumed that there is a WiFi connection between devices. Task Force type user, tanker vehicle and device match is pre-defined in this system so Server class can arrange messages to the related users.

Composition

The Connection package contains two main classes, Server class and Connector class. Server class receives and handles the connections from Task Force type users. Server class also sends message to the connected Task Force type users according to relation vehicle – user match. Connector class establishes connections to the Commanding type user. It also receives message from connected the Commanding type user and starts simulation of received message data.

Uses / Interactions

Both two type users interact with GUI. Server class is triggered by GUI. Server class also interacts with Connector class of Task Force type user. On the other hand Connector class has an interaction between Server class of the Commanding type user. Both Server class and Connector class extends Thread class to run asynchronous from other parts.

Resources

Simulated path data is sent and received by this package. Getting and forwarding of data is handled in this package. Moreover, XML Socket communication is handled in this package.

Processing

Simulated data is get from GUI package to the Server class. Server class sends message to the connected Connector classes with related Task Force type users which are pre-defined. It is assumed that there is Wi-Fi connection between devices. Only handling part of these connections is managed by Server class. Connector class sends received message to the GUI package to show user simulated data.

Interface / Exports

Connection package needs to use XML Socket connections to make connections between devices.

7.5 GUI

Classification

GUI is a package in this system. This package handles interaction between user and program. Shows lists to user and get and manage actions from user. GUI package consists of six classes, MainTab, Map, Tankers, Receivers, Simulation and MapRenderer.

Definition

It is stated that project will have a friendly user interface to interact with user effectively in Requirement Analysis Report. To achieve this goal GUI package will added to system. This package contains Tabbed Menu and Menu Tabs for main parts of projects.

Responsibilities

Main responsibility of GUI package is managing interaction between user and providing friendly graphical user interface to users. Map, Tankers, Receivers and Simulation classes are main tabs in the project. Map shows users lists of map data in the system to select visualized map. Tankers shows list of tanker type vehicles in the system that can deliver fuel to the receivers. Receivers class shows a list of receivers that receives fuel from tankers. Simulation shows simulation of paths. MainTab is responsible from controlling transitions between Menu Tabs and flow of program. MapRenderer is responsible from showing rendered map to the user and get interactions from user on map.

Constraints

In some Menu Tab user only can pass previous Menu Tabs to prevent errors in flow of program. MainTab parts enabling and disabling of sub-menus achieve this constraint.

Composition

Classes of this package are MainTab, Map, Tankers, Receivers, Simulation and MapRenderer. MainTab is the main tabbed menu that contains all of it. Map, Tankers, Receivers and Simulation are sub-menus. MapRenderer is rendered map which is only inside of Tankers, Receivers and Simulation sub-menus.

Uses / Interactions

GUI package mainly interacts with user. GUI gets user actions and show results of action. Passing other sub-menus, adding new tankers and receivers, actions on map and simulation are results of user actions. GUI also interacts with PatikaEngine and Connection packages.

Resources

Map data in the system is one of resources for GUI package. Map data is used to render map. Vehicles data in the system is another resource for GUI package. Vehicles data is used for creating lists in Tankers and Receivers. Path data from PatikaEngine is a resource to the GUI that is used in Simulation part to show shortest path.

Processing

When user opens program, MainTab will be seen. Firstly, Map sub-menu will be opened to show lists of map data in the system. When user selects one of them, that data will be used to render map in MapRenderer to show in other sub-menus. After rendering process,

Tankers sub-menu will be opened automatically and waiting from user to select one or more vehicles. Completion of selection process and clicking of next button will open Receivers menu to waiting from user to select receiver vehicles. When user clicks Simulate button simulation will be generated and will be showed to the user in Simulation sub-menu. User can play, pause and stop simulation and sends to the connected users via Connection sub-menu. User can go back any time but cannot go forward one more tab.

Interface / Exports

Tab Layout will implement to create menu system. MainTab will implement Tab Layout in Android SDK. Map sub-menu will implement Grid View in Android SDK. Tankers and Receivers sub-menus will implement Relative Layout Android SDK and these sub-menus contain lists that are implemented List View and Accordion View in Android SDK.

7.6 PathFinder

Classification

PathFinder is a package in this system. It contains PathFinderGenerator class that calculates shortest path with given mesh constraints.

Definition

In Requirement Analysis Report, is stated that shortest path between multiple points is the main purpose of this project. So PathFinder package contains implementation of shortest path algorithm for this project.

Responsibilities

Main responsibility of PathFinder package is calculating shortest path between multiple points. Moreover, finding shortest path between two points and between one – multiple points are another responsibilities of this package to achieve main goal.

Constraints

To make calculations this package have to get mesh and vehicle information as array. Mesh data and vehicle list have to be related to each other. In other words, each mesh data is related with specific vehicle.

Composition

PathFinder consists of only one class, PathFinderGenerator class. PathFinderGenerator consists of four main functions that calculate shortest paths, PathGenerate, Multiple-To-Multiple-Path, Multiple-To-One-Path and One-To-One-Path. PathFinderGenerator calculate shortest path in terms of start point number and target point number.

Uses / Interactions

PathFinder package only has an interaction with PatikaEngine class. It is called by PatikaEngine with necessary information to calculate shortest path. PathFinder returns Path data to the PatikeEngine.

Resources

Mesh data and vehicle list are main resources that coming from PatikaEngine. Mesh data and vehicle list must be related with each other.

Processing

PatikaEngine calls PathGenerate function in PathFinderGenerator with related information. PathGenerate calls Multiple-To-Multiple-Path, Multiple-To-One-Path, One-To-One-Path functions according to meshes and vehicles. PathGenerate returns calculated path data to the PatikaEngine to show user.

Interface / Exports

PathFinder package calculates shortest path with arrays of Mesh and Vehicle data types.

8. Libraries and Tools

8.1 JAVA

Java is a programming language, that is based on the principles of being object oriented, simple, robust and secure, architecture neutral and portable, high performance, interpreted, dynamic and threaded. It was originally developed by James Gosling at Sun Microsystems. We use Java as the main programming language of our project because; Java is the native programming language of Android SDK.

8.2 Eclipse IDE

Eclipse is a multi-language software development environment comprising an IDE and an extensible plug-in system. It is written mostly in Java and can be used to develop applications in Java. Eclipse began as an IBM Canada project. In November 2001, a consortium was formed to further the development of Eclipse as open-source software. In January 2004, the Eclipse Foundation was created. We use Eclipse IDE in our project because of the easy integration of Android SDK tools and emulators to this IDE.

8.3 Android SDK

Android is a software stack for mobile devices that includes an operating system, middleware, and key applications. The Android SDK provides the tools and libraries necessary to begin developing applications that run on Android-powered devices. Since we are also developing for Android based mobile platforms, we will be benefiting from Android SDK.

8.4 Android ADT

Android Development Tools (ADT) is a plug in for the Eclipse IDE that is designed to give the user an integrated environment in which to build Android applications. ADT offers some very useful extensions to Eclipse such as quickly setting up new Android projects, adding components based on the Android Framework API, opportunity to debug applications using the Android SDK tools, and exporting Android application package files in order to use our application on mobile systems. Since we are developing for Android based mobile platforms, we will also be benefiting from Android ADT.

8.5 Google USB Driver

The Google USB driver is a downloadable component for Windows developers, available for download from the AVD and SDK Manager. The Google USB Driver is only for Android Developer Phones. To send our Android application package files to the mobile devices, we need this driver.

8.6 Open GL ES

OpenGL ES is a cross-platform API for full-function 2D and 3D graphics on embedded systems including phones. It consists of well-defined subsets of desktop

OpenGL, creating a flexible and powerful low-level interface between software and graphics acceleration. The success of our application considerably lies on the quality of the visual aspects of it. Since this is a mobile application, we need powerful but low cost graphics. OpenGL ES will be the tool to provide this functionality to our project.

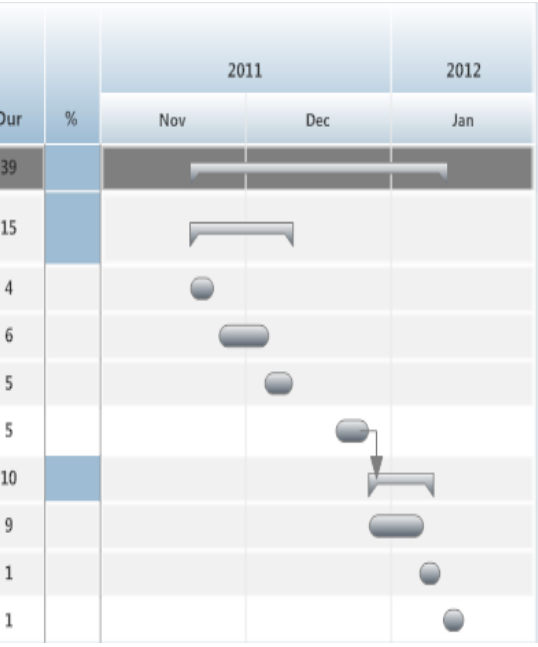
8.7 Android SDK API Level 8

The Android platform provides a framework API that applications can use to interact with the underlying Android system. Each released version of the Android platform includes updates to the Android application framework API that it delivers. We will be using level 8 frameworks API to present maximum device variability to users without losing needed development features.

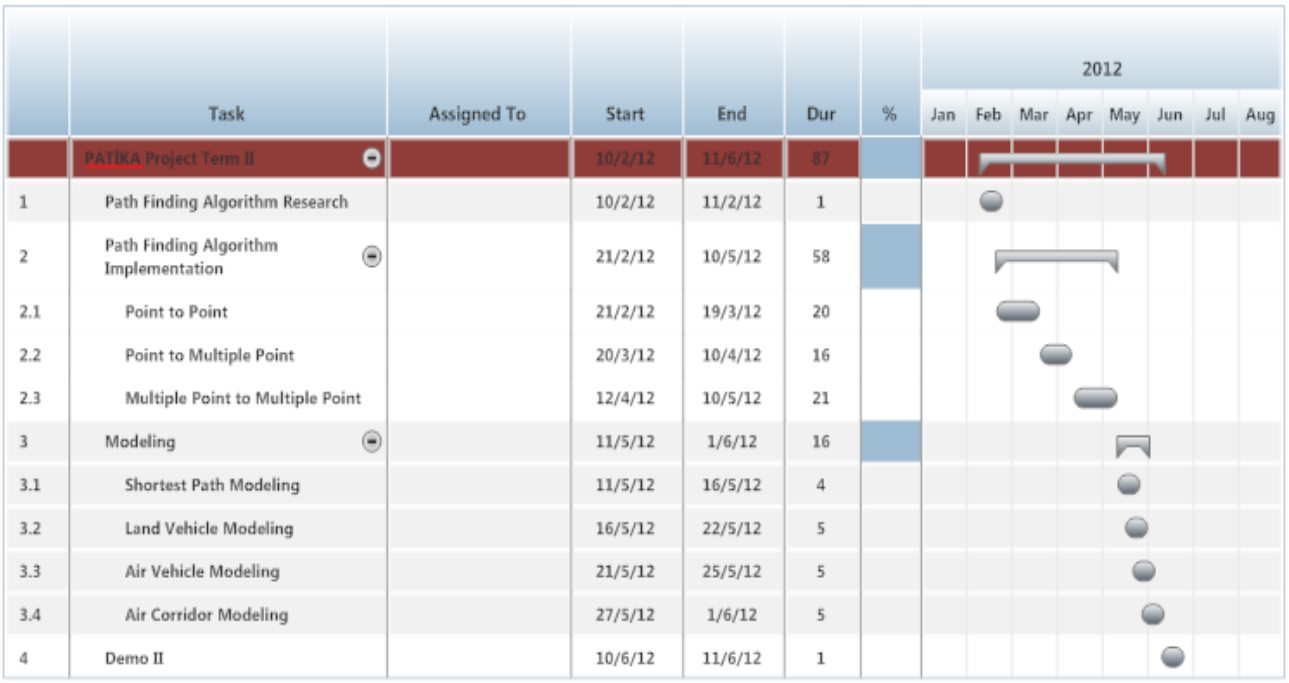
9. Time Planning

9.1 Term 1 Gantt Chart

	Task	Assigned To	Start	End	Dur	%
	Patika Project Term I		20/11/11	12/1/12	39	
1	Mobile Technology Implementation		20/11/11	10/12/11	15	
1.1	Touchpad		20/11/11	24/11/11	4	
1.2	Graphical User Menus		26/11/11	5/12/11	6	
1.3	Multi Touch		5/12/11	10/12/11	5	
2	Processing of DTED Data		20/12/11	26/12/11	5	
3	Visualization		27/12/11	9/1/12	10	
3.1	Map Visualization		27/12/11	7/1/12	9	
3.2	Air Corridor Visualization		7/1/12	9/1/12	1	
4	Demo 1		12/1/12	12/1/12	1	



9.2 Term 2 Gantt Chart



	Task	Assigned To	Start	End	Dur	%
	PATİKA Project Term II		10/2/12	11/6/12	87	
1	Path Finding Algorithm Research		10/2/12	11/2/12	1	
2	Path Finding Algorithm Implementation		21/2/12	10/5/12	58	
2.1	Point to Point		21/2/12	19/3/12	20	
2.2	Point to Multiple Point		20/3/12	10/4/12	16	
2.3	Multiple Point to Multiple Point		12/4/12	10/5/12	21	
3	Modeling		11/5/12	1/6/12	16	
3.1	Shortest Path Modeling		11/5/12	16/5/12	4	
3.2	Land Vehicle Modeling		16/5/12	22/5/12	5	
3.3	Air Vehicle Modeling		21/5/12	25/5/12	5	
3.4	Air Corridor Modeling		27/5/12	1/6/12	5	
4	Demo II		10/6/12	11/6/12	1	

10. Conclusion

In conclusion, Detailed Design Report for Patika has been written to explain the design and the architecture of the system in detail. Description, purpose and the scope of this project is also briefly discussed. Constraints and dependencies that can be faced are explained. The tools and the libraries that will be used during development the project are given. Data flow models, class diagrams, interface features, entity relationship diagrams, possible use cases are also given within the document. A planning of the project spanning two semesters is included. Detailed algorithmic description of components and data is also given.