



**Middle East Technical University**  
**Department of Computer Engineering**

**Computer Engineering Design 1**

Fall 2011

**Initial Design Report for PATİKA Project**

**Striders of the Modern World**

**Levent Oral**

**Pınar Yılmaz**

**Bahar Şevket**

**Gözde Özcel**

# TABLE OF CONTENTS

<b>1 INTRODUCTION.....</b>	<b>4</b>
1.1 PROBLEM DEFINITION .....	4
1.2 PURPOSE .....	4
1.3 SCOPE .....	5
1.4 OVERVIEW .....	5
1.5 DEFINITIONS, ACRONYMS AND ABBREVIATIONS.....	5
1.6 REFERENCES .....	6
<b>2 SYSTEM OVERVIEW .....</b>	<b>6</b>
<b>3 DESIGN CONSIDERATIONS .....</b>	<b>7</b>
3.1 DESIGN ASSUMPTIONS, DEPENDENCIES AND CONSTRAINTS .....	7
3.2 DESIGN GOALS AND GUIDELINES .....	8
<b>4 DATA DESIGN .....</b>	<b>8</b>
4.1 DATA DESCRIPTION.....	8
4.2 DATA DICTIONARY .....	8
4.2.1 <i>Model Package</i> .....	8
4.2.1.1 Class BaseVehicle .....	8
4.2.1.2 Class LandVehicle .....	11
4.2.1.3 Class AirVehicle .....	12
4.2.1.4 Class Path .....	13
4.2.1.5 Class Point .....	14
4.2.1.6 Class AirCorridor .....	14
4.2.2 <i>Engine Package</i> .....	15
4.2.2.1 Class PatikaEngine .....	15
4.2.2.2 Class Mesh.....	16
4.2.3 <i>Factories Package</i> .....	16
4.2.3.1 Class MapFactory .....	16
4.2.3.2 Class PathFactory .....	16
4.2.3.3 Class VehicleFactory .....	17
4.2.4 <i>Connection Package</i> .....	17
4.2.4.1 Class Server .....	17
4.2.4.2 Class Connector .....	18
4.2.5 <i>GUI Package</i> .....	18
4.2.5.1 Class MainTab.....	19
4.2.5.2 Class Map .....	19
4.2.5.3 Class Tankers .....	19
4.2.5.4 Class Receivers .....	19
4.2.5.5 Class Simulation .....	20
4.2.5.6 Class MapRenderer .....	21
4.2.6 <i>PathFinder Package</i> .....	21
4.2.6.1 PathGenerator Class .....	21
<b>5 SYSTEM ARCHITECTURE .....</b>	<b>22</b>
5.1 ARCHITECTURAL DESIGN .....	22
5.2 DESCRIPTION OF COMPONENTS .....	23
5.2.1 <i>Model</i> .....	24
5.2.1.1 Processing narrative for component Model .....	25
5.2.1.2 Interface Description for component Model .....	25
5.2.1.3 Processing Detail for component Model.....	25
5.2.1.4 Dynamic Behavior for component Model.....	26
5.2.2 <i>Engine</i> .....	26
5.2.2.1 Processing narrative for component <i>Engine</i> .....	26

5.2.2.2 Interface Description for component <i>Engine</i> .....	27
5.2.2.3 Processing Detail for component <i>Engine</i> .....	27
5.2.2.4 Dynamic Behavior for component <i>Engine</i> .....	28
5.2.3 <i>Factories</i> .....	29
5.2.3.1 Processing narrative for component <i>Factories</i> .....	29
5.2.3.2 Interface Description for component <i>Factories</i> .....	30
5.2.3.3 Processing Detail for component <i>Factories</i> .....	30
5.2.3.4 Dynamic Behavior for component <i>Factories</i> .....	30
5.2.4 <i>Connection</i> .....	30
5.2.4.1 Processing narrative for component <i>Connection</i> .....	31
5.2.4.2 Interface Description for component <i>Connection</i> .....	31
5.2.4.3 Processing Detail for component <i>Connection</i> .....	31
5.2.4.4 Dynamic Behavior for component <i>Connection</i> .....	32
5.2.5 <i>GUI</i> .....	32
5.2.5.1 Processing narrative for component <i>GUI</i> .....	33
5.2.5.2 Interface Description for component <i>GUI</i> .....	33
5.2.5.3 Processing Detail for component <i>GUI</i> .....	33
5.2.5.4 Dynamic Behavior for component <i>GUI</i> .....	34
5.2.6 <i>PathFinder</i> .....	34
5.2.6.1 Processing narrative for component <i>PathFinder</i> .....	34
5.2.6.2 Interface Description for component <i>PathFinder</i> .....	35
5.2.6.3 Processing Detail for component <i>PathFinder</i> .....	35
<b>6 USER INTERFACE DESIGN .....</b>	<b>35</b>
6.1 SCREENSHOTS .....	35
6.1.1 <i>Map Selection Screen</i> .....	35
6.1.2 <i>Carrier Selection Screen</i> .....	36
6.1.3 <i>Receiver Selection Screen</i> .....	37
6.1.4 <i>Simulation Screen</i> .....	37
<b>7 LIBRARIES AND TOOLS .....</b>	<b>37</b>
7.1 JAVA .....	37
7.2 ECLIPSE IDE .....	37
7.3 ANDROID SDK .....	37
7.4 ANDROID ADT .....	38
7.5 GOOGLE USB DRIVER .....	38
7.6 OPENGL ES .....	38
7.7 ANDROID SDK API LEVEL 8.....	38
<b>8 TIME PLANNING .....</b>	<b>39</b>
8.1 TERM 1 GANTT CHART .....	39
8.2 TERM 2 GANTT CHART .....	39
<b>9 CONCLUSION.....</b>	<b>40</b>

# **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]<http://en.wikipedia.org/wiki/Pathfinding>  
Pathfinding algorithms and common usage

[2]<http://forum.9kgames.com/default.aspx?g=posts&t=128>  
A star pathfinding for Android, concepts and implementation

[3][http://researchspace.csir.co.za/dspace/bitstream/10204/4200/1/Leenen\\_2010.pdf](http://researchspace.csir.co.za/dspace/bitstream/10204/4200/1/Leenen_2010.pdf)  
A Constraint-based Solver for the Military Unit Path Finding Problem

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

[5] 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

[6]<http://www.gliffy.com>  
Web-based diagram editor

[7]<http://developer.android.com/guide/topics/graphics/opengl.html>  
Web page that usage of opengl ES n android is clearly explained.

[8][http://en.wikipedia.org/wiki/Use\\_case\\_diagram](http://en.wikipedia.org/wiki/Use_case_diagram)  
Use case diagrams

[9]<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.
- 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.

### 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 user. 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 applicaiton. 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.

- **Integer 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 constraints 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.
- **Vector3 latitude:** Latitude of the point object is kept in this field in Vector3 type.
- **Vector3 longitude:** Longitude 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 necessary 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 GroupLayout 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

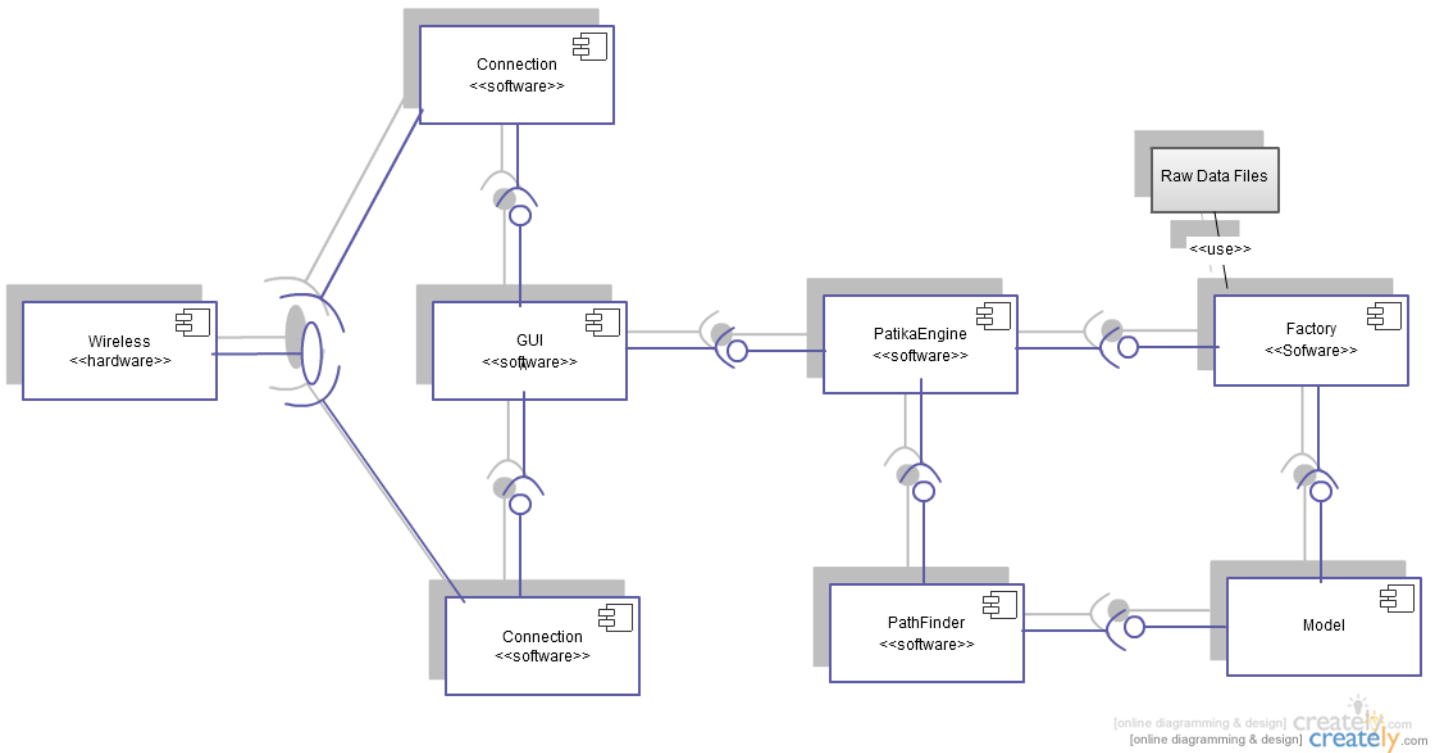
PATİKA has six main components; GUI, Engine, Factory, Connections, Model and Pathfinder.

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 5.1**

## 5.2 Description of Components

The package diagram included below intends to show object-oriented nature of the basic software components of the whole PATIKA 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.

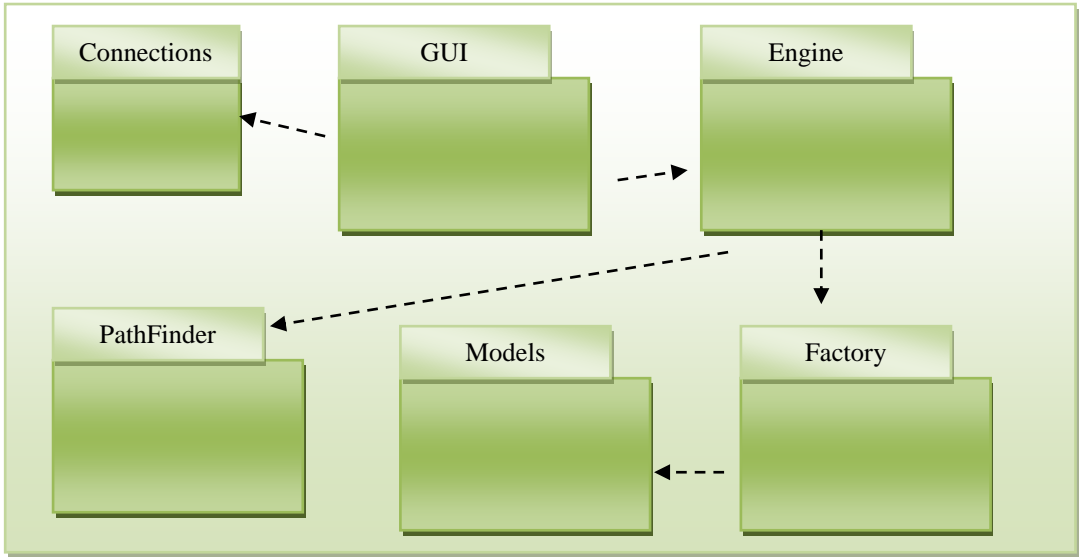
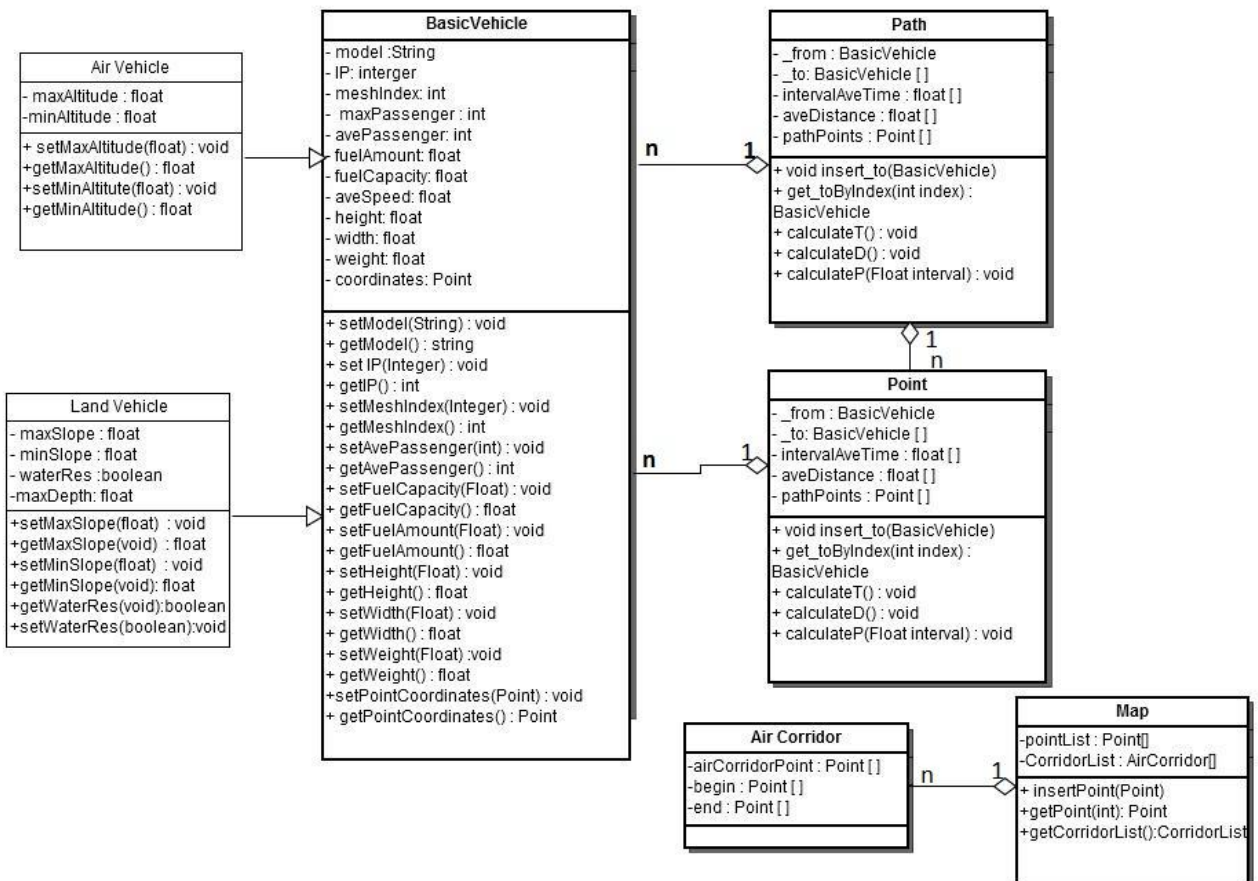


Figure 5.2

### 5.2.1 Model

#### Diagram for Model



### **5.2.1.1 Processing narrative for component Model**

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.

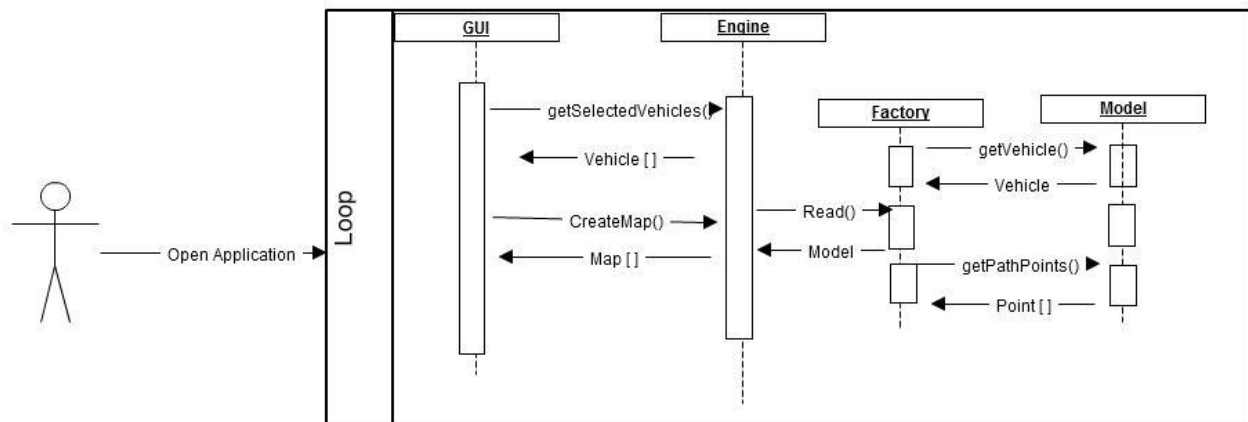
### **5.2.1.2 Interface Description for component Model**

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 Model**

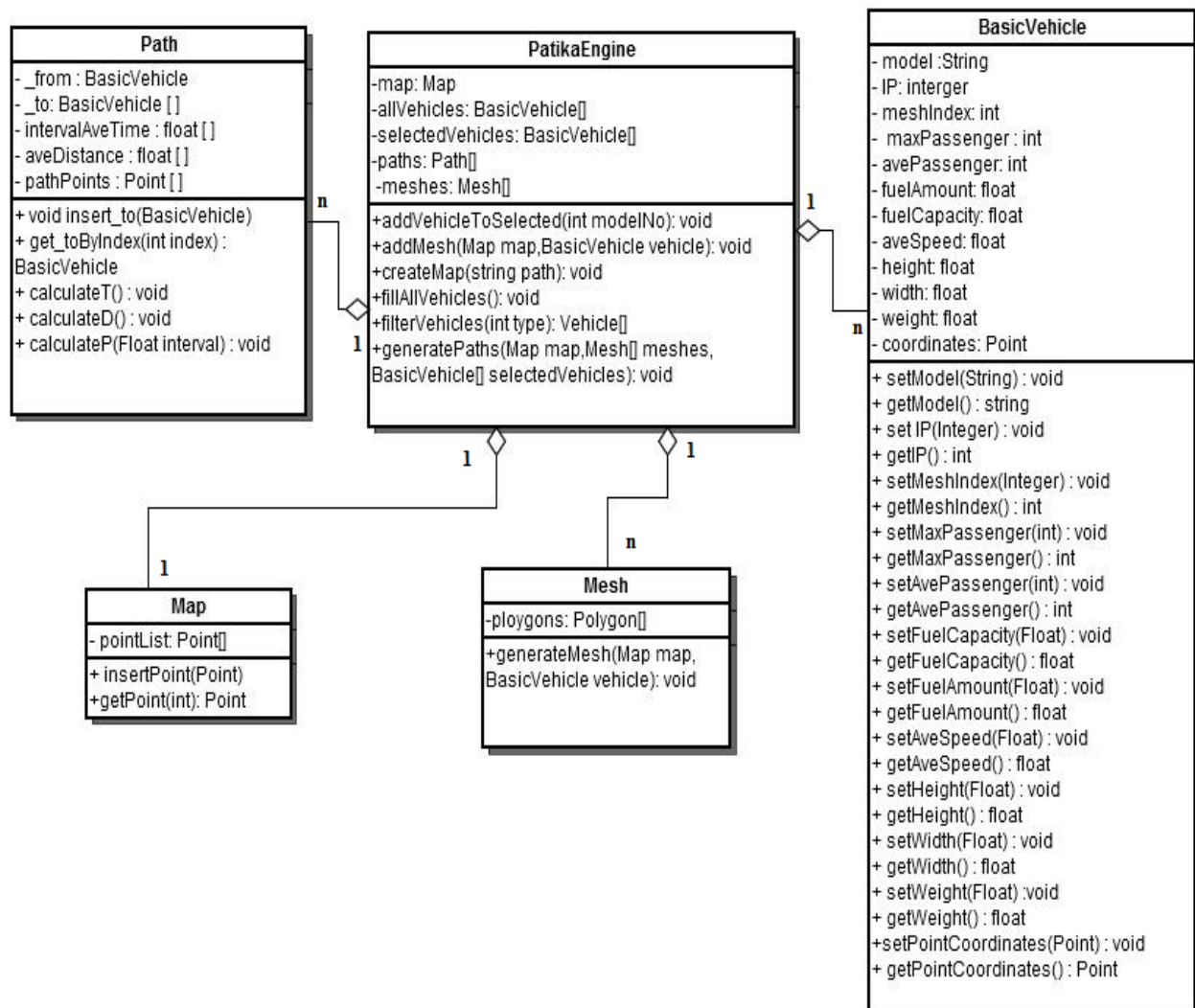
Since there are many classes and many operations performed of these classes in this component, the processing details will be explained briefly for main classes. Factory components are used for generating meaningful instances of classes in Model. The MapFactory read map data from file, according to each 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 Behaviour for Component Model



### 5.2.2 PatikaEngine

#### Diagram for Component Engine



### **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

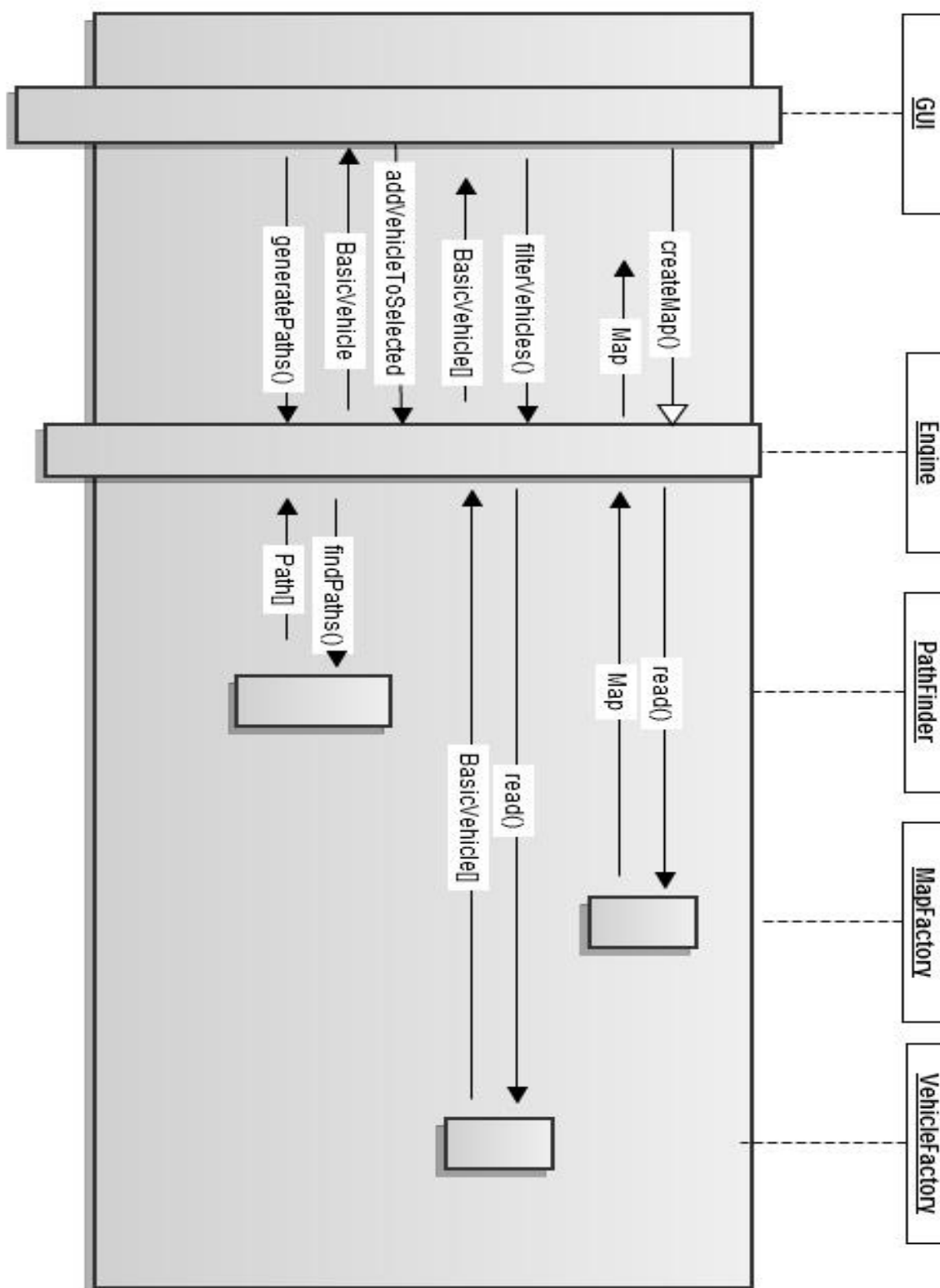


Figure 5.2.2.4

## 5.2.3 Factories

### Diagram for Component Factories

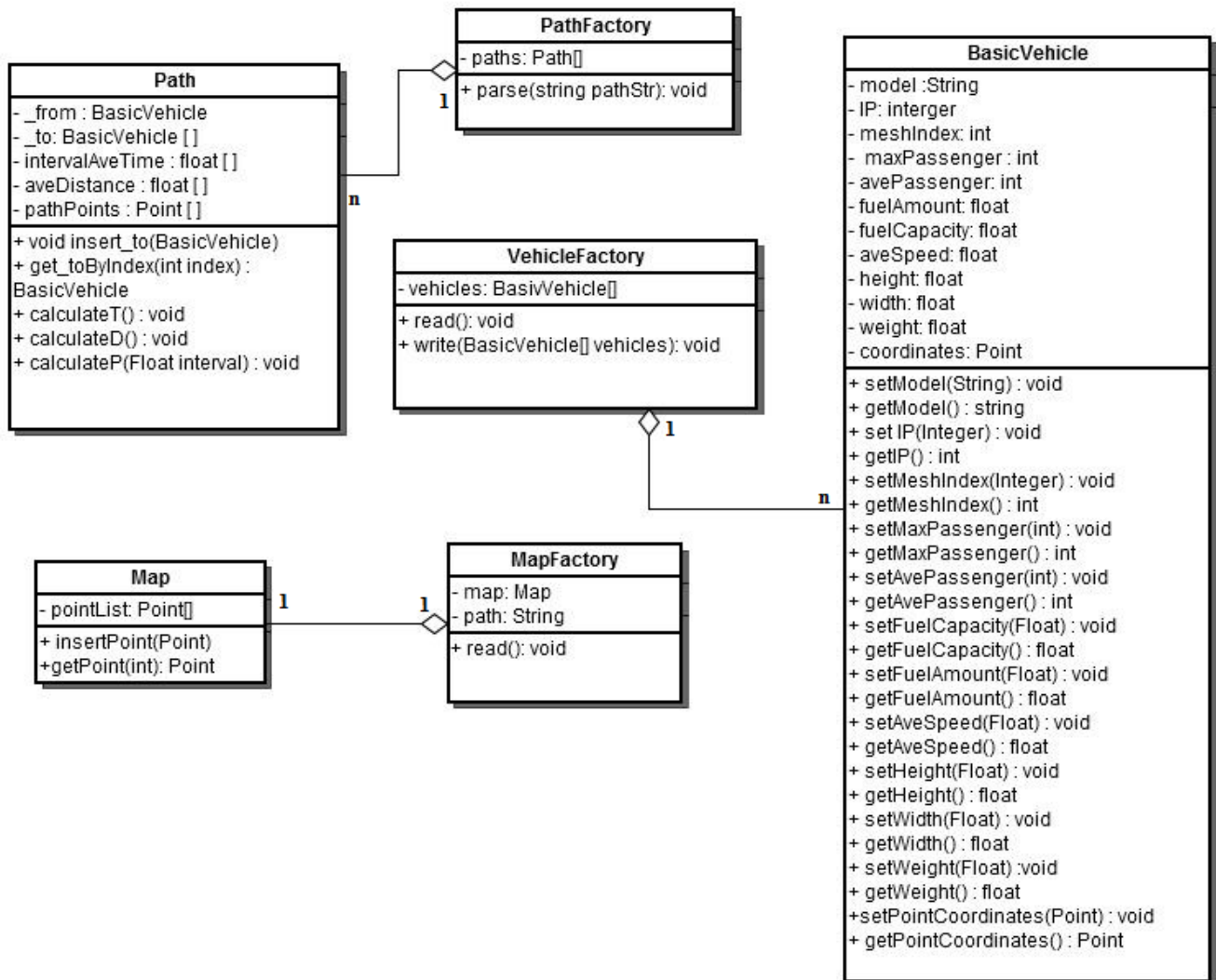


Figure 5.2.3

#### 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

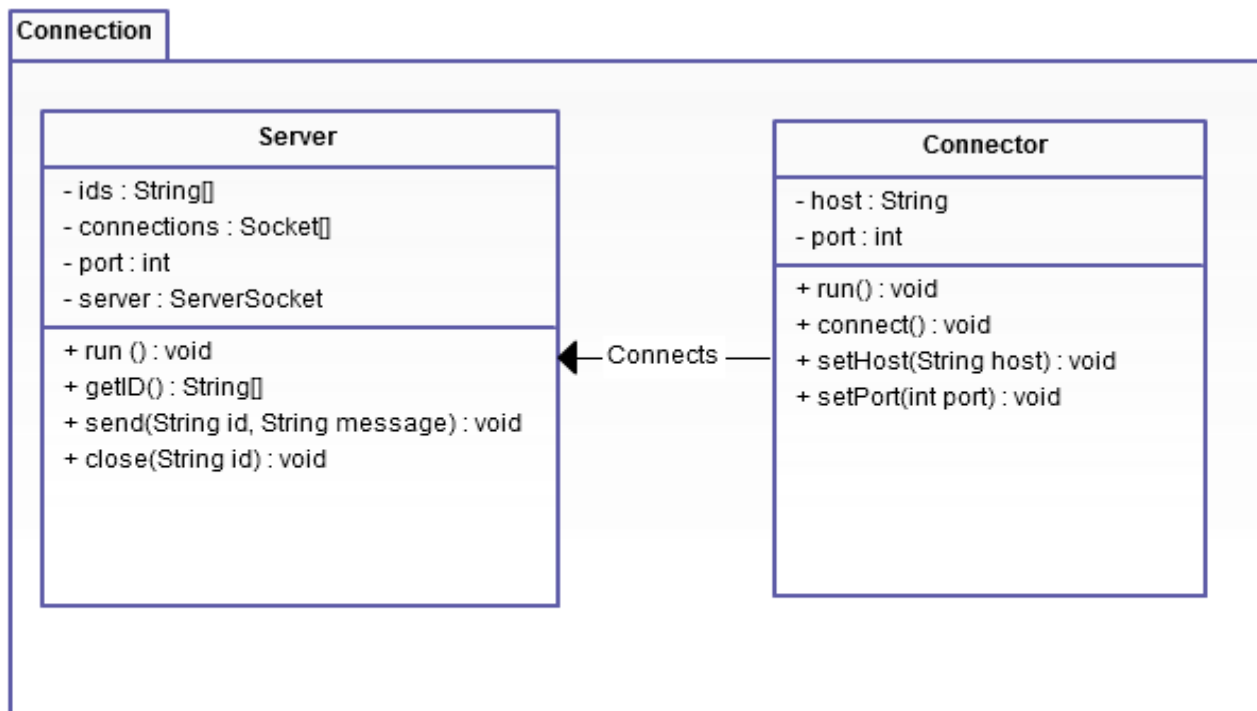
Factories component will need three inputs. First one is the file path that is needed while reading the map file. The second one is a string variable that contains path information that will be used when parsing and generating the Path [] instance. Third input, which is an array of BasicVehicles, is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



#### **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 Behaviour Component Connection

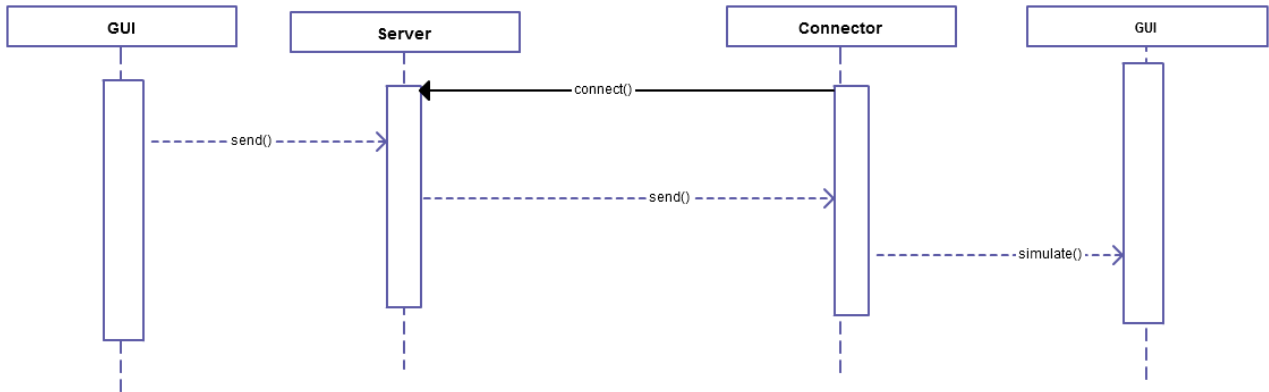
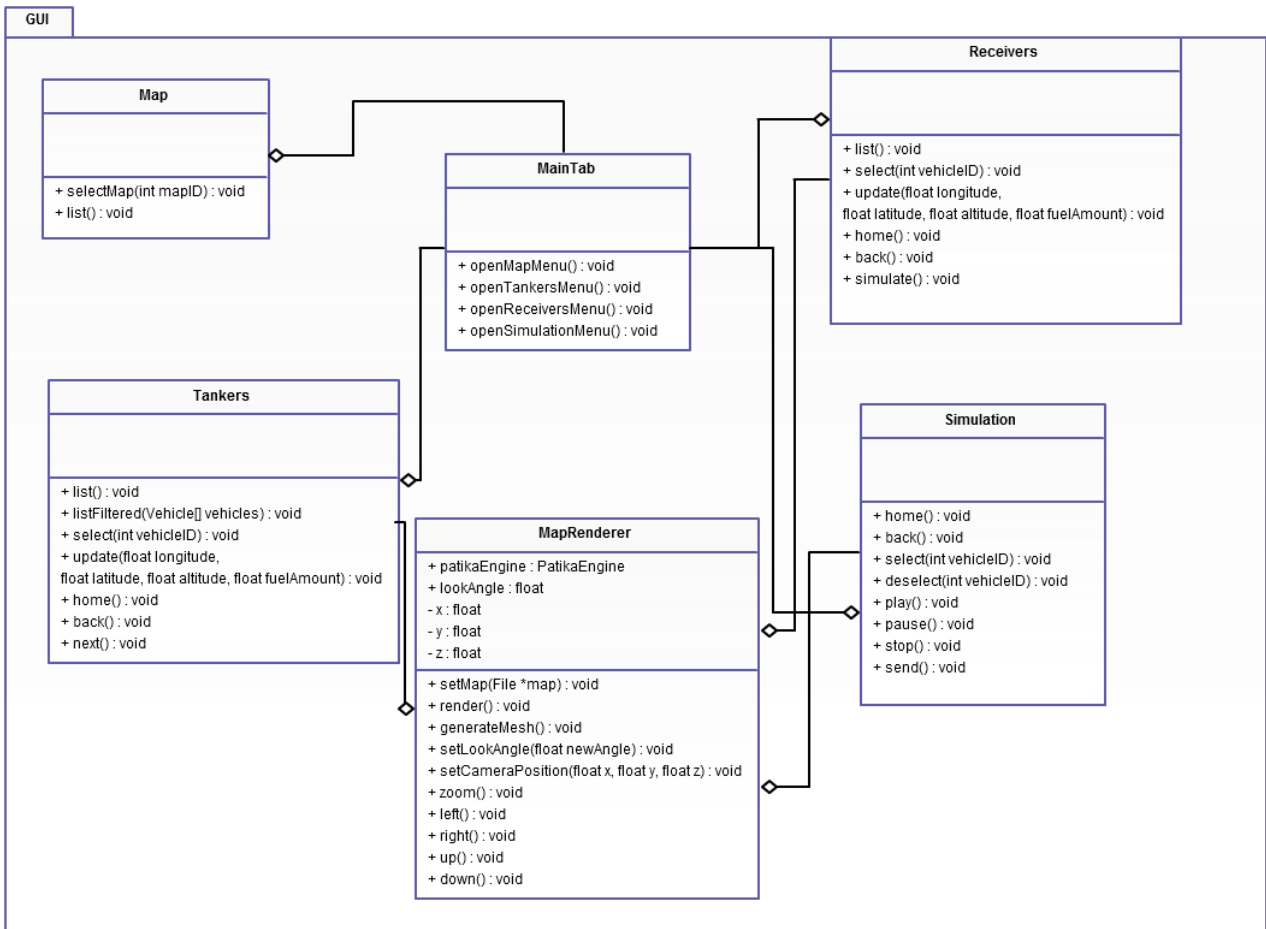


Figure 5.2.4.4

### 5.2.5 GUI

#### Diagram for Component GUI



### **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, 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 CommandDevice.

### **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 Connection

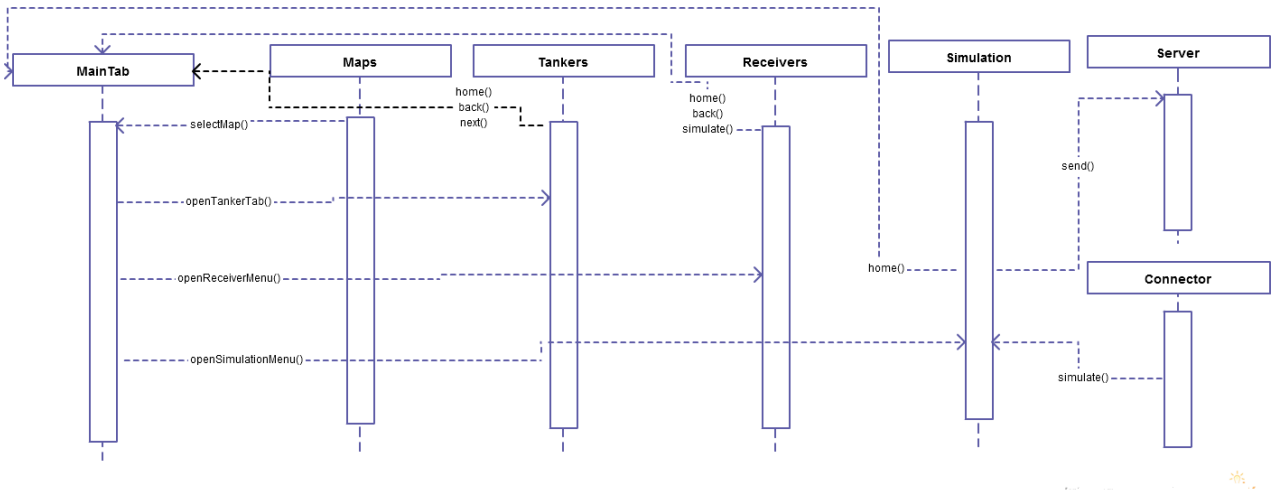
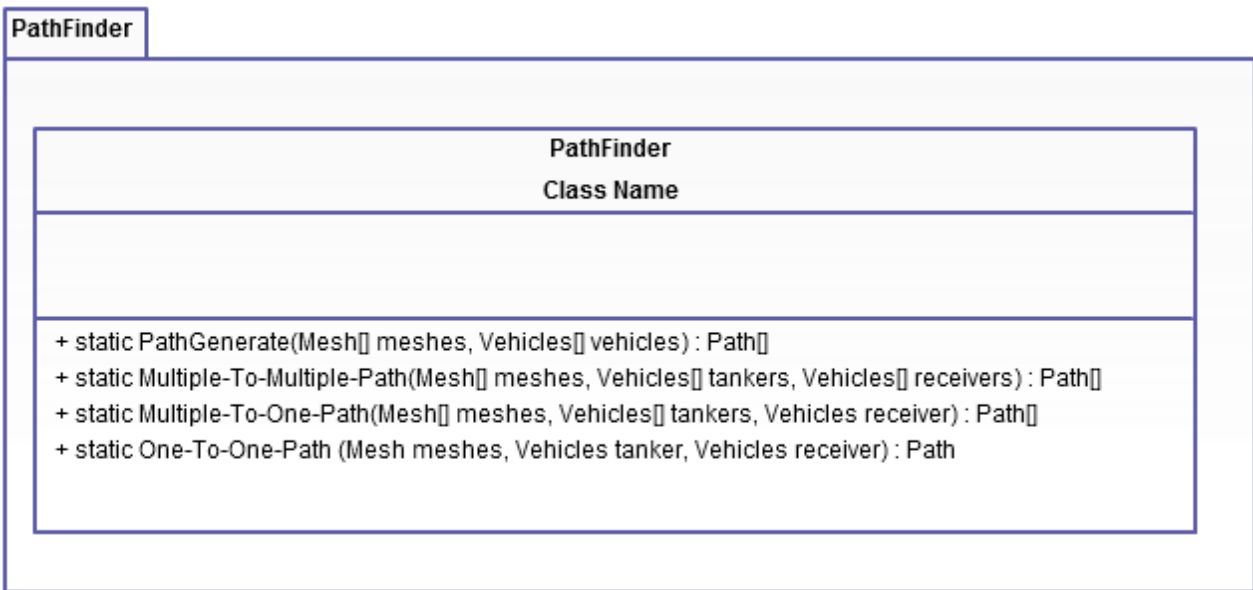


Figure 5.2.5.4

### 5.2.6 PathFinder



#### 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.

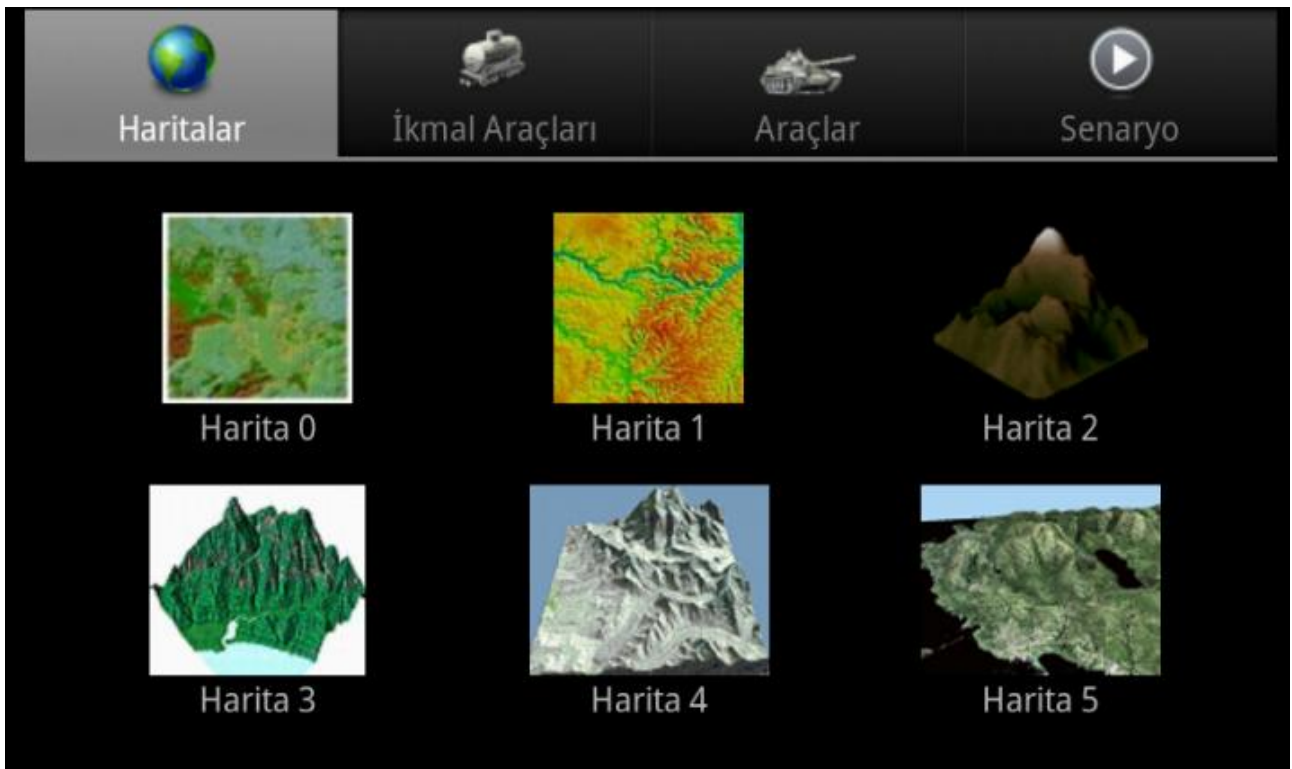
### 5.2.6.3 PathFinder Processing Detail

- 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.

## 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 Receiver Vehicle Selection Menu**

It will be the same as the fuel carrier selection menu, but instead of carrier vehicles, receiver 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. Libraries and Tools**

### **7.1 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.

### **7.2 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.

### **7.3 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.

## **7.4 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.

## **7.5 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.

## **7.6 Open GL ES**

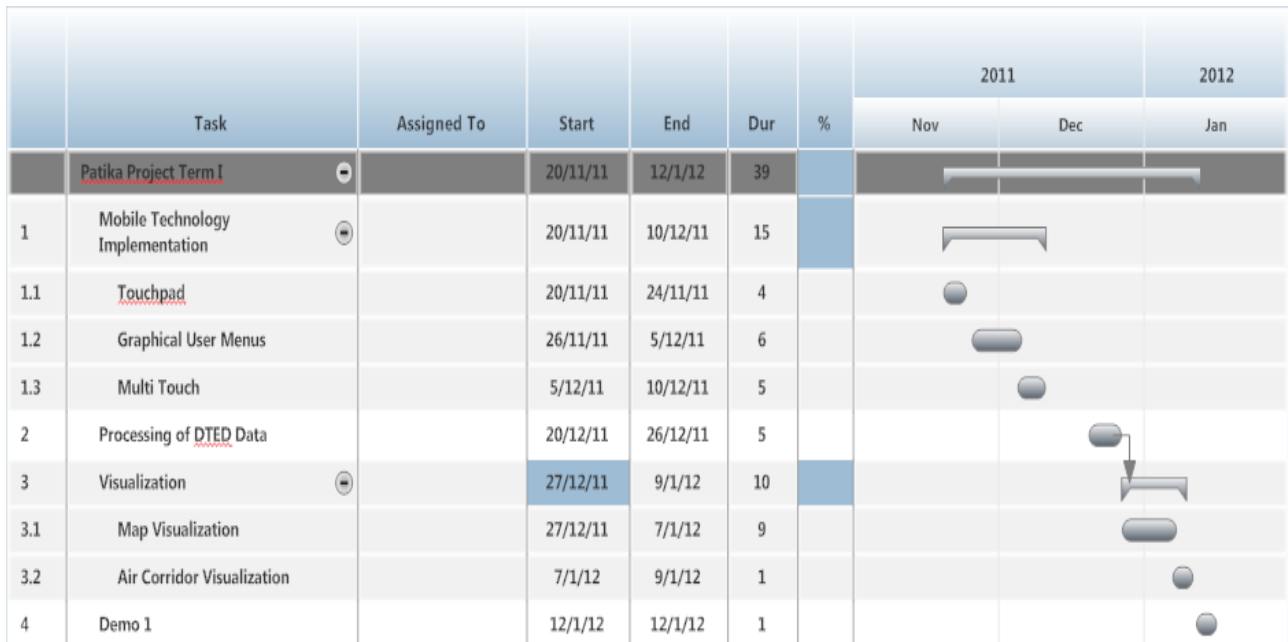
OpenGL ES is a cross-platform API for full-function 2D and 3D graphics on embedded systems including phones. It consists of well-defined subsets of desktop OpenGL, creating a flexible and powerful low-level interface between software and graphics acceleration. The success of our application considerably 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.

## **7.7 Android SDK API Level 8**

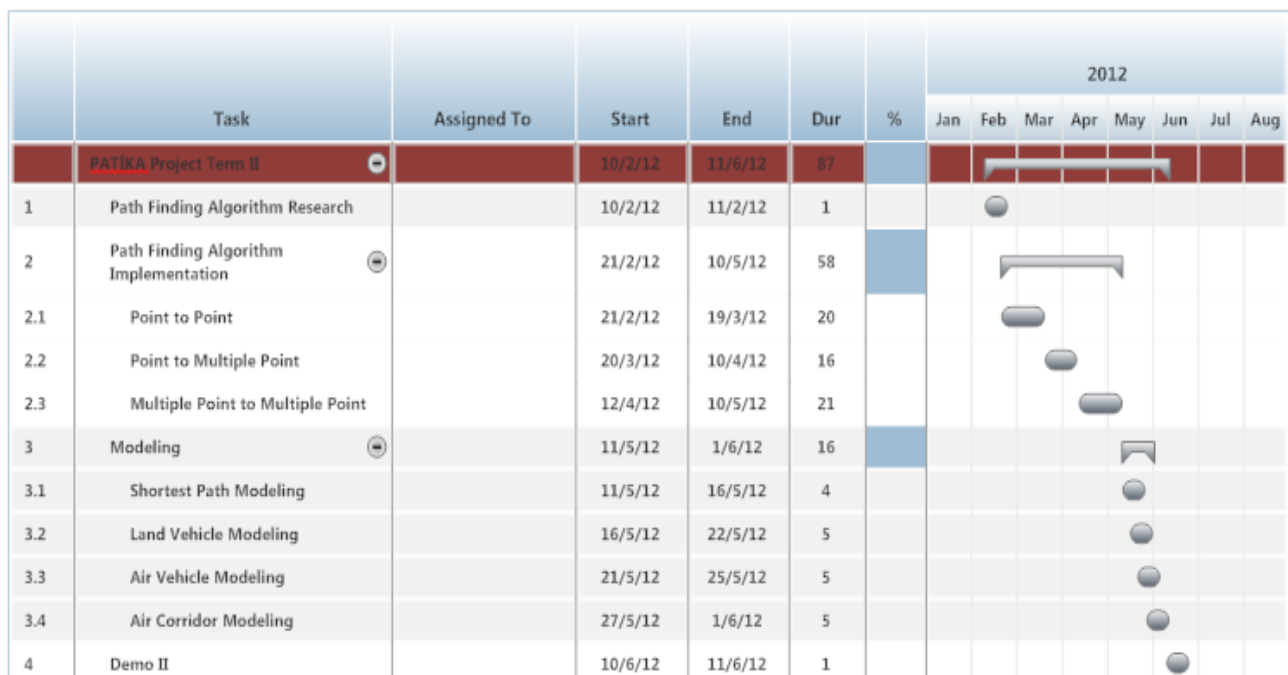
The Android platform provides a framework API that applications can use to interact with the underlying Android system. Each released version of the Android platform includes updates to the Android application framework API that it delivers. We will be using level 8 frameworks API to present maximum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

## 8. Time Planning

### 8.1 Term 1 Gantt Chart



### 8.2 Term 2 Gantt Chart



## 9. Conclusion

In conclusion, Initial Design Report for Patika has been written to explain the design and the architecture of the system. 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 will be given in the Detailed Design Report.