M.E.T.U

Computer Engineering Department
CENG 491

DETAILED DESIGN REPORT

TEAM BABYLON:

M. Tuğberk İŞYAPAR - 1502996
Arif S. ÖZÇELİK - 1428044
Okan T. TURSUN - 1445550
Ö. Anıl TÜRKKAN - 1449750

Assistant: Fatih GÖKÇE

January, 17 2010
# Table of Contents

## 1 Introduction

### 3.1 Purpose of the Document

### 3.2 Problem Definition

### 3.3 Project Goals

## 2 Design Constraints

### 4.1 Time Constraints

### 4.2 Resource Constraints

### 4.3 Performance Constraints

### 4.4 Software Constraints

### 4.5 Hardware Constraints

## 3 Modelling of the System

### 5.1 Overview of The System

#### 5.1.1 Vehicles

#### 5.1.2 Roadside Equipment

#### 5.1.3 Coordinators

### 5.2 GUI Design

#### 5.2.1 Menu Bar

#### 5.2.2 Controls

#### 5.2.3 Map

### 5.3 Network Message Format

#### 5.3.1 Performance of the Network

## 4 Technologies Used in the Project

### 6.1 Network Structure

#### 6.1.1 Possible Issues Regarding the Implementation

### 6.2 GPS NMEA

#### 6.2.1 Data Link Layer

#### 6.2.2 General Sentence Format

### 6.3 GPSD

### 6.4 JAVA Swing API

#### 6.4.1 Foundations (7)

### 6.5 OpenMap API
DETAILED DESIGN REPORT

6.5.1 Software Interfaces Provided by OpenMap ........................................ 18
6.6 GrooveNet ................................................................................................. 20
   6.6.1 General Properties ............................................................................. 21
   6.6.2 GUI .................................................................................................... 21
   6.6.3 Internal Properties ............................................................................. 22
7 SYSTEM DESIGN .......................................................................................... 25
   7.1 Component Diagram ............................................................................ 25
   7.2 Package Diagram .................................................................................. 26
   7.3 ZigBee Communication Package ......................................................... 27
      7.3.1 Data Dictionary for the ZigBee Communication Package ............. 28
   7.4 GPS Package .......................................................................................... 30
      7.4.1 Data Dictionary for the GPS Package ........................................... 30
   7.5 Vehicles Package ................................................................................... 31
      7.5.1 Data Dictionary for Vehicles Package ........................................... 32
   7.6 Road Side Facilities Package ............................................................... 35
      7.6.1 Data Dictionary for Road Side Facilities Package ......................... 36
   7.7 Traffic Signs Package ............................................................................ 38
      7.7.1 Data Dictionary for Traffic Signs Package ..................................... 38
   7.8 Map Package .......................................................................................... 40
      7.8.1 Data Dictionary for Map Package .................................................... 41
   7.9 Sequence Diagrams ............................................................................... 43
   7.10 State And Activity Diagram ................................................................. 48
8 SCHEDULE .................................................................................................... 50
   8.1 Tasks Accomplished ................................................................................ 50
   8.2 Division Of Labor ................................................................................... 51
   8.3 Gantt Chart ............................................................................................. 51
9 Bibliography .................................................................................................. 52

Figure 1: Roles of Nodes in the Vehicular Ad Hoc Network ......................... 8
Figure 2: HAMMUR’ABI GUI Design .............................................................. 10
Figure 3: XBEE PRO ZNET 2.5 Transceiver Hardware ............................... 13
Figure 4: GrooveNet Interface ...................................................................... 22
Figure 5: GrooveNet Classes ...................................................................... 22
Figure 6: Component Diagram ..................................................................... 25
Figure 7: Package Diagram .......................................................................... 26
Figure 8: Class Diagrams for ZIGBEE Communication Package ........................................... 27
Figure 9: Class Diagrams for GPS Package ........................................................................... 30
Figure 10: Class Diagrams for Vehicles Package ................................................................. 32
Figure 11: Class Diagrams for Road Side Facilities Package ............................................... 35
Figure 12: Class Diagrams for Traffic Signs Package ............................................................ 38
Figure 13: Class Diagrams for Map Package ........................................................................ 40
Figure 14: Sequence Diagram in Case of No Accident ........................................................... 45
Figure 15: Sequence Diagram in case of an accident ............................................................ 46
Figure 16: Sequence Diagram for Setting a Zoom Level Via User Interface ......................... 47
Figure 17: Sequence Diagram for Finding Shortest Path Between Current Location and Destination ................................................................................................................. 47
Figure 18: State Diagram ...................................................................................................... 48
Figure 19: Activity Diagram ................................................................................................ 49
Figure 20: Gantt Chart ........................................................................................................... 51

Table 1: Vehicular Information Transfer Protocol ................................................................... 12
Table 2: XBEE PRO ZNET 2.5 Technical Specifications ....................................................... 14
Table 3: Data Link Layer ......................................................................................................... 16
Table 4: Message Class .......................................................................................................... 28
Table 5: Receiver Class .......................................................................................................... 29
Table 6: Config Class ............................................................................................................. 29
Table 7: Transmitter Class ..................................................................................................... 30
Table 8: GPSDATA Class ...................................................................................................... 30
Table 9: VEHICLE Class ........................................................................................................ 32
Table 10: Fire Fighter, Ambulance, Police and Automobile Classes ....................................... 33
Table 11: GPSWATCHER Class ............................................................................................ 34
Table 12: LOCATION Class ................................................................................................... 34
Table 13: SensorWatcher Class ............................................................................................ 34
Table 14: INFORMER Class ................................................................................................. 35
Table 15: ROADSIDEFACILITY Class .................................................................................. 36
Table 16: Hospital, Fire House, Police Station Classes .......................................................... 37
Table 17: EMERGENCYWATCHER Class ......................................................................... 37
Table 18: Location Class ....................................................................................................... 37
Table 19: Informer Class ........................................................................................................ 38
Table 20: TrafficSign Class ................................................................................................... 39
Table 21: Location Class ....................................................................................................... 39
Table 22: Informer Class ........................................................................................................ 40
Table 23: Map Class .............................................................................................................. 41
Table 24: MapEntity Class .................................................................................................... 41
Table 25: Location Class ....................................................................................................... 42
Table 26: MapRefresher Class .............................................................................................. 43
Table 27: GUI Interface ........................................................................................................ 43
2 REVISION HISTORY

- OpenStreetMap is not used anymore. Instead OpenMap will be used.
- OpenMap is discussed in detail.
- GrooveNet is chosen as the simulation tool.
- Since Map Interface is done by OpenMap, JAVA2D API is no longer used.
- State and Activity diagrams are added.
- Two new sequence diagrams are added.
- Package diagram is updated by adding OpenMap API Package.
3 INTRODUCTION

3.1 PURPOSE OF THE DOCUMENT

This report provides the necessary definitions to conceptualize and further formalize the design of the software, of which its requirements and functionalities were summarized in the previous requirements analysis report. The aim is to provide a guide to a design that could be easily implemented by any designer reading this report.

3.2 PROBLEM DEFINITION

Traffic and transportation systems on roads get more and more complex every day. Although there are equipments and instruments providing extensively detailed information for naval and air traffic, vehicles on land lack these kind of advantages. Despite that automobiles are equipped with sophisticated devices today; these devices are only designed to provide information for the vehicle they are working on. They do not share information with other vehicles that could be valuable in some decision making processes for drivers. Information regarding traffic density, accidents, nearby police and ambulances, speed of other nearby vehicles can be very useful most of the time. Today’s technology is more than enough to measure this information and present to driver. Using this information, complexity of traffic can be reduced. The Hammur’Abi Vehicular Ad-Hoc Network system is based on gathering this kind of information and making the driver fully aware of his/her location and destination.

3.3 PROJECT GOALS

The goals of introducing such a product will be;

- To provide drivers with the knowledge of traffic conditions, like density or accidents, so that they would have the opportunity to adjust their course.
- To assist passengers in case of emergency with finding police or ambulances.
- To help emergency vehicles like ambulances and fire fighters to travel quick and safely by informing vehicles in front of them of their existence.
- To lower the risks of accidents by displaying information about other vehicles’ speed and reminding drivers of certain speed limits and other constraints.
- To send hospitals and police stations information about recent accidents.
• To provide automobile drivers with information regarding possible car parking spaces in especially big and crowded metropolis.

4 DESIGN CONSTRAINTS

4.1 TIME CONSTRAINTS

This project is a senior student project, given by the department of Computer Engineering. So the schedule and timing is determined and strict. After this report a prototype must be presented by mid-January, 2010. The implementation should be fully done by June, 2010.

4.2 RESOURCE CONSTRAINTS

Although the Istanbul Metropolitan Municipality (1) is thinking of using a traffic control system based on digital camera recordings and using Global Positioning System (GPS) data, in our country the idea of implementing a traffic control system the way we propose it is relatively new. There are some examples that are not the same but nevertheless related to what we think of producing, in different countries. However some of these are still on paper, under discussion or not widely used.

4.3 PERFORMANCE CONSTRAINTS

The system will depend heavily on the modules which will be used for vehicle-vehicle and vehicle-road side equipment communication. No database infrastructure will be included. Input will be retrieved from the wireless communication device and interpreted. Most important measure of performance will be how much time it will take the system to react to sudden changes in network structure. Since project does not involve designing a new hardware and network protocol for project’s purpose, the system will be limited by capacity of current hardware implementations.

4.4 SOFTWARE CONSTRAINTS

• Most, if not all of the code will be developed using Java Development Toolkit and Java Runtime Environment.

• The platforms to develop code are NetBeans and Eclipse.
• The open-source software OpenMap will be used for drawing a map on the user interface.
• For simulation purposes ‘GrooveNet’ Software will be used.

4.5 HARDWARE CONSTRAINTS

• The system will run on a laptop kind of device.
• Since no hardware component is supplied, constraints of GPS and Zigbee Devices are not known.

5 MODELLING OF THE SYSTEM

5.1 OVERVIEW OF THE SYSTEM

The system will have a network structure which supports dynamic routing and mesh topology. Nodes in the network and their roles are depicted in the figure below:

![Diagram of roles of nodes in the vehicular ad hoc network](image)

**FIGURE 1: ROLES OF NODES IN THE VEHICULAR AD HOC NETWORK**

There are three main roles that a node can take. It can be a coordinator, roadside equipment or a vehicle. Coordinator is the node which starts the network. There is exactly one coordinator in each network and it is stationary. Roadside equipments are the nodes which can transmit and receive information. They provide valuable information to vehicles (such as speed limit on a road, or location of pharmacy etc.). Types of roadside equipment are illustrated in the class diagram later in the report. Vehicles are nodes in traffic flow. They can be civilian cars or other vehicles which posses special types such as ambulances, fire trucks and police cars.
5.1.1 VEHICLES

Vehicles can have different types in the system such as fire fighter, ambulance, police car and automobile (details are given in class diagrams section). Each vehicle introduces its identity to the system in periodic broadcasts along with its coordinates. Sharing vehicle type information enables drivers to be aware of them and act accordingly. There can be zero or more vehicle existent in the network at any given time. System will support dynamic entry and exit of vehicles to the network.

5.1.2 ROADSIDE EQUIPMENT

Roadside equipments are stationary nodes which will share information that is valuable for vehicles in the network. Information sharing can be one way (in case of traffic signs for example) or two ways (equipments placed in hospitals which share coordinates of hospital and watch the network for possible emergency signals). There can be zero or more roadside equipment in the network. Roadside equipment is stationary and it is assumed that roadside equipment is always in range of another roadside equipment or coordinator. If no vehicle is present in the system, this ensures that roadside equipments are still connected to the network to deliver information.

5.1.3 COORDINATORS

Coordinators are the stationary nodes that start the network. There is exactly one coordinator in each network. There is a limit on the number of nodes in each network depending on the wireless communication hardware and protocol. The constraints of the network hardware will be discussed at following chapters. For initial implementation, it is assumed that the capacity of network will be sufficient to support required node capacity but as the required capacity increases in future use, it may be required to use more improved network hardware and protocols.

5.2 GUI DESIGN

This section illustrates basics of GUI that will be implemented in Hammur’Abi Vehicular Ad-Hoc network. Please note that since the project is in early development stage, the layout proposed in this section may be improved later. Software interfaces of OpenMap which are triggered by GUI are discussed in section 6.5 OpenMap API. The GUI will consist of three main parts: Menu Bar, Controls and Map.
5.2.1 MENU BAR

The menu bar will provide access to menus for adjusting main settings of the system.

**Connection menu:** User will be able to control connectivity via this menu such as network connection/disconnection, opening network selection window.

**Map menu:** User will be able to load new maps to the system for offline use or organize the map files that are already loaded to the system.

**Configure menu:** With this menu, user will be able to control system configuration such as display settings, connection properties (e.g. automatic connection to any network in range) etc.
Find menu: This menu will provide a shortcut for user to find nearest hospital, police station, pharmacy, car park etc. The map will point to the found location. If desired location is not found in range, user will be informed from the message space in controls part of the GUI.

Filter menu: User will be able to define filter rules depending on which entities he/she wants to see in the map.

5.2.2 CONTROLS

The controls part of the GUI will have widgets for functions of the system which can be frequently used. For example, there will be buttons for centering own vehicle in the map, centering given coordinates or centering coordinator of the network (the device which starts the network), resetting a previously set destination on map. There are two other components of controls part which are zooming control and message space from which user obtains valuable information and warnings provided by the system.

5.2.3 MAP

Map will display information about streets, roads as well as entities such as other vehicles around driver. Map will represent different entities with different icons and colors. For instance, in the sample design provided above, ordinary automobiles are represented by black boxes whereas own vehicle is indicated by a red square. Police car is indicated with a blue “P”, coordinator is represented by green capital “C” letter, hospital by red “H” and so on. In order not to clutter the map with lots of data, detailed information about an entity (such as speed and heading of vehicles) can be displayed when pointer moves on them. Panning can be achieved by clicking and dragging map, similar to other map tools. Destination can be entered to the system by right click on the map and system will draw a route which indicates the streets the driver should pass through. In the example given above, there is an increase in traffic density on the road between Monument and Bank. In such cases, if the shortest path includes this road, the system can inform the driver about the congestion via message space and find another route without high traffic density.

5.3 NETWORK MESSAGE FORMAT

The communication between nodes in the network is accomplished by passing messages between nodes. The message format that will be used in the system is modified version of Vehicular Information Transfer Protocol (VITP) specification (2).
TABLE 1: VEHICULAR INFORMATION TRANSFER PROTOCOL

<table>
<thead>
<tr>
<th>VITP message syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line 1: METHOD &lt;uri&gt; VITP/&lt;version_number&gt;</td>
</tr>
<tr>
<td>Line 2: Target: [rd_id-dest,seg_id-dest]</td>
</tr>
<tr>
<td>Line 3: From: [rd_id-src,seg_id-src] with &lt;speed&gt;</td>
</tr>
<tr>
<td>Line 4: Time: &lt;current_time&gt;</td>
</tr>
<tr>
<td>Line 5: Expires: &lt;expiration_time&gt;</td>
</tr>
<tr>
<td>Line 6: Cache-Control: &lt;directive&gt;</td>
</tr>
<tr>
<td>Line 7: msgID: &lt;unique_key&gt;</td>
</tr>
<tr>
<td>Line 8: Content-Length: &lt;number of bytes&gt;</td>
</tr>
<tr>
<td>Line 9: CRLF</td>
</tr>
<tr>
<td>Line 10: &lt;message body&gt;</td>
</tr>
</tbody>
</table>

In Line1 METHOD can be GET (query) or POST (reply). The rest of this line can be omitted in the messages since message format is not in standard VITP format. Data in Lines 2-3 contain road ID and segment ID data. Instead of this, in our system messages will contain geographical coordinates obtained from GPS hardware in message body. In addition to this, 16-bit or 64-bit network addresses will be used as source and target addresses. For stationary nodes such as coordinator and roadside equipment <speed> is set to default value 0. Line 6 cache-control is not implemented in the system so it can be omitted. A similar functionality as expiration time in Line 5 is implemented by setting maximum number of hops so it will be omitted in the system. Other information such as vehicle types will be passed in message body.

5.3.1 PERFORMANCE OF THE NETWORK

At any given time the amount of traffic that can be carried over a given node is limited by ZigBee hardware. In Xbee PRO Znet 2.5 datasheet the speed of data transmission is given as 250 Kbps. But this specification does not take mobility of nodes and corresponding data overhead in the network. In such cases, there can be increase in package drop rates and latencies in the network depending on the rate of change (i.e. speed of vehicles in the network). The real performance of the system under these conditions can be found by simulation only. There are some simulation results discussed by Dikaikos, Iqbal, Nadeem and Iftode (3). But all these results are obtained on a network based on IEEE 802.11.
6 TECHNOLOGIES USED IN THE PROJECT

6.1 NETWORK STRUCTURE

Vehicular ad hoc network (VANET) is an application area of broader topic, Mobile Ad Hoc Networks (MANET). MANET is a type of mesh network in which every node acts as a router and data packets are “hopping” through nodes till they reach their destinations. However, although the topology of MANETs is same as mesh networks, they have to take into account the changes in the connections due to movement of nodes. There are various wireless communication standards in the literature that are used in VANET prototypes some of which are:

- UMTS Terrestrial Radio Access with Time Division Duplexing (UTRA-TDD)
- IEEE 802.11 (WiFi)
- IEEE 802.16 (WiMAX)
- HiperLAN/2
- Bluetooth
- IRA
- Cellular and satellite technologies
- ZigBee and ZigBee PRO

Among these technologies, ZigBee PRO will be used as the basis for the system. ZigBee is a wireless mesh networking standard whose development started in 1998. ZigBee was proposed as an alternative to other wireless technologies where low energy consumption and self organizing network structure were required. In 2007, an enhanced ZigBee specification called ZigBee PRO was introduced. The implementation of the project will be realized using XBee PRO Znet 2.5 hardware which is based on latest ZigBee PRO standard.

FIGURE 3: XBEE PRO ZNET 2.5 TRANSCEIVER HARDWARE

XBee PRO Znet 2.5 has the following specifications:
TABLE 2: XBEE PRO ZNET 2.5 TECHNICAL SPECIFICATIONS

<table>
<thead>
<tr>
<th>Data Rate</th>
<th>250 kbps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range (Outdoor)</td>
<td>1.6 km</td>
</tr>
<tr>
<td>Serial Data Interface</td>
<td>3.3V CMOS UART</td>
</tr>
<tr>
<td>Serial Data Rate</td>
<td>1200 bps – 1 Mbps</td>
</tr>
</tbody>
</table>

The ZigBee hardware whose specs are given above can be connected to PC over RS232 interface. USB connection is also possible with RS232 to USB hardware converters. For both configuration of ZigBee hardware, the given data rate and range are appropriate to use this hardware as a part of VANET. However there are some issues regarding implementation which will be discussed at the end of this section.

There are three types of nodes in a ZigBee network:

1. Coordinator: is the node which starts the network. Coordinator is responsible for setting the PAN ID which is unique id number of the network and an unoccupied RF communication channel. After that coordinator is ready to permit joining of other nodes to the network either as routers or end devices. After network is started, coordinators can send data to nodes in the network or route data packets to other nodes depending on packet’s destination. There is exactly one coordinator in each PAN.

2. Router: There can be zero or more routers in the network. They are responsible for routing data packets. It is also possible for routers to send data packets generated by them. Routers can permit joining of nodes to network by assigning them network addresses. Vehicles in a VANET are suitable to be routers in ZigBee network since they can both transmit their own data and route other vehicles’ data.

3. End device: End devices are designed to nodes where energy consumption is critical. End devices are in a low power consuming idle state when they do not transmit data. End devices cannot route packets of other nodes and are not suitable for use in VANETs.

All devices which join a ZigBee network have two types of addresses which identify themselves. First one is a 16-bit address which changes every time the node disconnects from the network and rejoins. This is like IP address in TCP/IP and assigned by a coordinator or router in the network to the device. Second address is 64-bit hardware address and is permanent i.e. it does not change upon disconnection and rejoin.

Data can be sent in two modes in ZigBee network. Either it can be sent as unicast transmission where the recipient is a single node, or it can be sent as broadcast transmission in which every single node in the network is the recipient. 16-bit temporary address or 64-bit
permanent address of the recipient can be used to send data packet. When data packet is to be sent over the network, if network route to the destination is not known, first a route discovery is performed. Ad Hoc On-demand Distance Vector routing algorithm is used to find the optimal path in ZigBee networks.

6.1.1 POSSIBLE ISSUES REGARDING THE IMPLEMENTATION

Every time a node disconnects and rejoins network or a node in the network cannot connect to the desired destination, route discovery is repeated. This brings additional network traffic if the connectivity of nodes changes periodically in a ZigBee network as in case of VANETs. We did not experimented with the ZigBee network hardware yet and hence we do not know if ZigBee network adapts this connectivity changes fast enough due to movement of cars. With small number of slowly moving nodes, it should not be a problem but if the system will be used in dense and rapidly changing network conditions such as highway traffic, ZigBee specifications might be inadequate.

It is stated in the datasheet of XBee PRO that in rare circumstances destination might receive data packet from the source; however, source might stay unaware of that. In that case, same data is retransmitted and destination can receive same data again. There is not any internal filtering in ZigBee network for that. The software system should do the filtering of same data.

Datasheet of XBee PRO discourages frequent use of broadcasting since packet “hops” over all nodes in the network which generates large amount of network traffic. However, broadcast messages can be frequently used especially when vehicles are in emergency mode and while vehicles are informing environment about their type and status. Such information addresses everyone in the network and a maximum hop limit can be put to prevent generation of excessive network traffic.

6.2 GPS NMEA

Since the proposed system requires a GPS device, NMEA 0183 (NMEA) will be used during communication between the device and the system. NMEA is combined electrical and data specification for communication between GPS receivers (4). The standard uses a simple ASCII, serial communications protocol that defines how data is transmitted in a "sentence" from one "talker" to multiple "listeners" at a time.
6.2.1 DATA LINK LAYER

TABLE 3: DATA LINK LAYER

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baud rate</td>
<td>4,800</td>
</tr>
<tr>
<td>Data bits</td>
<td>8</td>
</tr>
<tr>
<td>Parity</td>
<td>None</td>
</tr>
<tr>
<td>Stop bits</td>
<td>1</td>
</tr>
<tr>
<td>Handshake</td>
<td>None</td>
</tr>
</tbody>
</table>

6.2.2 GENERAL SENTENCE FORMAT

All data is transmitted in the form of sentences. Only printable ASCII characters are allowed, plus CR (Carriage return) and LF (line feed). Each sentence starts with a "$" sign and ends with <CR><LF>. There are three basic kinds of sentences: talker sentences, proprietary sentences and query sentences.

6.2.2.1 TALKER SENTENCES

The general format is:

$ttsss,d1,d2,...,<CR><LF>

The first two letters following $ is the talker identifier and next three are sentence identifiers. Then a number of comma separated data fields exist in the message. The message is terminated by a carriage return/line feed.

Example: $HCHDM,238,M<CR><LF>

HC: Talker is a magnetic compass.
HDM: Magnetic heading message follows
238: Heading value
M: Heading value is magnetic

6.2.2.2 PROPRIETARY SENTENCES

The standard allows individual manufacturers to define proprietary sentence formats. These sentences start with "$P", then a 3 letter manufacturer ID, followed by whatever data the manufacturer wishes, following the general format of the standard sentences.

6.2.2.3 QUERY SENTENCES

A listener can request a particular sentence from a talker by following format:

$tillQ,sss,<CR><LF>
DETAILED DESIGN REPORT

First two characters are again the identifiers of the talker and the next two characters are the talker identifier of the device being queried (listener). The fifth character is always a ‘Q’ defining the message as a query. The “sss” contains the representation of the sentence being requested.

Example: $CCGPQ,GGA<CR><LF>

The CC (computer) requests the “GGA” sentence from the GPS device. Then the GPS will transmit this sentence every second until a different query arrives.

6.3 GPSD

gpsd is a service daemon that monitors one or more GPSes or AIS receivers attached to a host computer through serial or USB ports, making all data on the location/course/velocity of the sensors available to be queried on TCP port 2947 of the host computer. With gpsd, multiple location-aware client applications (such as navigational and war driving software) can share access to receivers without contention or loss of data. Also, gpsd responds to queries with a format that is substantially easier to parse than the NMEA 0183 emitted by most GPSes. The gpsd distribution includes a linkable C service library, a C++ wrapper class, and a Python module that developers of gpsd-aware applications can use to encapsulate all communication with gpsd (6).

Since a great number of NMEA sentences are not open to public, it will be very complicated to write a parser for the GPS device. As a result, we will use this library a daemon between GPS device and the computer.

6.4 JAVA SWING API

Java Swing is an API or providing a graphical user interface (GUI) for Java programs. This API will be used for creating user interface.

6.4.1 FOUNDATIONS (7)

1. Swing is platform independent.
2. Swing is a highly partitioned architecture, which allows for the "plugging" of various custom implementations of specified framework interfaces: Users can provide their own custom implementation(s) of these components to override the default implementations.
3. Swing is a component-based framework. The distinction between objects and components is a fairly subtle point: concisely, a component is a well-behaved object with a known/specified characteristic pattern of behavior.

4. Given the programmatic rendering model of the Swing framework, fine control over the details of rendering of a component is possible in Swing.

5. Swing's heavy reliance on runtime mechanisms and indirect composition patterns allows it to respond at runtime to fundamental changes in its settings.

6. Swing's configurability is a result of a choice not to use the native host OS's GUI controls for displaying itself. Swing "paints" its controls programatically through the use of Java 2D APIs, rather than calling into a native user interface toolkit. Thus, a Swing component does not have a corresponding native OS GUI component, and is free to render itself in any way that is possible with the underlying graphics APIs.

### 6.5 OPENMAP API

OpenMap is a Java toolkit which is used in building applications for displaying and manipulating geographical data. OpenMap is provided by BBNT Solutions LLC and is open source and freely available on the internet (8).

OpenMap provides Java Swing components that can be used to draw maps and graphical objects on maps. Data is represented and rendered in terms of layers. Layers are Java Swing components and OpenMap Developer Hints states that their rendering are taken care of automatically.

There are numerous classes and methods provided by OpenMap. Important architecture components of OpenMap that provide interfaces to our implementation are described in the next section.

### 6.5.1 SOFTWARE INTERFACES PROVIDED BY OPENMAP

Most of the widgets that the user interacts with in the GUI window are provided as Swing components by OpenMap. This section provides a summary of methods bound to GUI components as well as software interfaces of important OpenMap classes.

**MapBean:** MapBean is a Swing component which extends java.awt.Container class. It can be added to Java windows like other user interface components. MapBean is the main map window component of OpenMap.
Interface: void setLayers(LayerEvent evt)
Description: LayerListener interface method. Modifies layers of map.
Triggered By: Program launch, load of new shapefile.

Interface: void zoom(ZoomEvent evt)
Description: Zooms the map.
Triggered By: User changing zoom level from GUI.

Interface: void setCenter(float lat, float lon)
Description: Sets the center of the map.
Triggered By: User clicking “Center Own”, “Center Coordinator” or “Center coordinates” button in GUI.

Interface: void pan(PanEvent evt)
Description: Handles incoming PanEvents.
Triggered By: NavigatePanel generating new PanEvents.

**Layer:** Layers are used to display maps and other graphical information. Layers may contain streets, centers of cities, boundaries of countries etc. Layers are added to objects of MapBean class. Each layer can be enabled and disabled independently.

Interface: public ShapeLayer(java.lang.String shapeFileName)
Description: Constructor which generates new shapelyer from file.
Triggered By: User loading new map using Map menu, JFileChooser passes resulting filename to the constructor.

Interface: public void setVisible(boolean show) Description: Sets visibility of the layer.
Triggered By: User changing visibility from Configure menu.
MouseEvents: When mouse enters/exits window, clicks an entity, an object of MouseEvent class is generated. This object describes type of the event and the coordinates where the event has occurred.

Interface: MapMouseEvent: LatLonPoint getLatLon()
Description: Returns the latitude and longitude of the point where mouse event occurred.
Triggered By: User clicking on map.

ToolPanel: ToolPanel is a class extending JToolBar. It provides widgets to interact with the map.

Interface: NavigatePanel class
Description: Provides arrows to pan the map in 8 directions.
Triggered By: User clicking buttons in GUI.

Interface: ZoomPanel class
Description: Provides buttons to zoom in and out in the map.
Triggered By: User clicking buttons in GUI.

OMGraphic: Base class of graphics. The classes extending OMGraphic are used to represent entities on the map. In our implementation, other cars and places will be represented by OMGraphic objects.

Interface: void select()
Description: Set the selected attribute to true, and sets the color to the select color.
Triggered By: MouseEvent objects.

6.6 GROOVENET
GrooveNet will be the simulator for the project. As stated before, GrooveNet is the most advanced simulator in V.A.N.E.T systems. But there were some problems with the modification rights of the simulator. However, the problem is solved after contacting the developer.

### 6.6.1 GENERAL PROPERTIES

Here are some general properties of the simulator without any modification (9):

- GrooveNet is a modular event based simulator.
- GrooveNet simulates GPS and supports multiple vehicle models over a variety of network models.
- The graphical interface makes it easy to generate the simulations.
- GrooveNet supports three types of simulated nodes: vehicles which are capable of multi-hopping data over one or more DSRC channels, fixed infrastructure nodes and mobile gateways capable of vehicle-to-vehicle and vehicle-to-infrastructure communication.
- GrooveNet supports multiple message types such as GPS messages, which are broadcast periodically to inform neighbors of a vehicle's current position, and vehicle emergency and warning event messages with priorities. Multiple rebroadcast policies have been implemented to investigate the broadcast storm problem.
- GrooveNet is able to support hybrid (i.e. communication between simulated vehicles and real vehicles on the road) simulations where simulated vehicle position, direction and messages are broadcast over the cellular interface from one or more infrastructure nodes. Real vehicles communicate with only those simulated vehicles which are within its transmission range.
- GrooveNet is coded with C++ and works under Linux.

### 6.6.2 GUI

During simulation default user interface will be:
6.6.3 **INTERNAL PROPERTIES**

In this chapter, the structure of the simulator will be discussed.

#### 6.6.3.1 **MODEL CLASS INHERITENCE HIERARCHY**

---

**FIGURE 4: GROOVENET INTERFACE**

**FIGURE 5: GROOVENET CLASSES**
Several types of vehicles are used in GrooveNet: the infrastructure node, a GPS-powered vehicle, a network-powered vehicle, a simulated vehicle, and a simulated vehicle not linked to the map database (unconstrained).

### 6.6.3.2 COMMUNICATION MODELS

There are three basic layers used by all vehicles for communication. The physical layer, which is modeled by the CarPhysModel class and derived classes, represents the physical hardware and the communication medium itself (e.g. wireless). The link layer, which is modeled by the CarLinkModel class and derived classes, represents the filter protocol used to discriminate between relevant and irrelevant network traffic; the link layer passes relevant traffic up to the application. The communication layer, which is modeled by the CarCommModel class and derived classes, represents the rebroadcast policy used by a vehicle to forward network traffic. Each of the derived classes is discussed below:

- **CarPhysModel**
  - **SimplePhysModel** – performs simple check to enforce communication range; supports multiple communication channels on the same receiver.
  - **CollisionPhysModel** – same as SimplePhysModel, but includes collision modeling.
  - **MultiPhysModel** – models multiple receivers and uses different physical layer models for messages from vehicles and messages from infrastructure nodes; especially useful for mobile gateways.

- **CarLinkModel**
  - **SimpleLinkModel** – verifies that the packet has not been previously received and that the receiver lies within that packet’s geographic region of relevance

- **CarCommModel**
  - **SimpleCommModel** – rebroadcasts packets at a specific rate; supports jittering
  - **AdaptiveCommModel** – same as above, but implements several advanced features: first/fast rebroadcasting, distance-based rate throttling, load-based rate throttling, and location-based broadcast suppression; each of these features can be enabled or disabled
  - **GrooveCommModel** – (not yet implemented at time of release) a placeholder for the GrooveNet Adaptive Broadcast Model.
6.6.3.2.2 MOBILITY MODELS

There are several types of mobility models. First, the infrastructure node uses no mobility model, assuming that it is stationary. The infrastructure node’s position can be specified in the InfrastructureNodeModel class itself. The GPSModel and NetModel classes derive their mobility from external sources (a Global Positioning System and a network, respectively). The SimUnconstrainedModel class uses special unconstrained mobility models. These models, such as that implemented in the RandomWaypointModel class, specify a starting location and a method for wandering in an unconstrained two-dimensional space. Finally, the SimModel class is the most sophisticated, using two types of models to simulate motion: a mobility model for executing motion and a trip model for planning routes. The derived classes for each of these are described below:

- SimUnconstrainedMobilityModel
  - RandomWaypointModel – wander at a random speed and direction continuously
- SimMobilityModel
  - FixedMobilityModel – remain at a fixed position/do not move
  - StreetSpeedModel – vehicle always moves at the speed limit
  - UniformSpeedModel – vehicle moves at some speed uniformly distributed about the speed limit; the user specifies the lower and upper bounds relative to the speed limit in either absolute or percentage-based values
  - CarFollowingModel – vehicle will not exceed the speed of the vehicle in front, vehicle will use a different mobility model (user-specified) when vehicle is the leader

- SimTripModel
  - RandomWalkModel – vehicle will randomly choose where to go whenever vehicle approaches an intersection
  - DjikstraTripModel – vehicle plans the shortest route from the source to the destination location (possibly visiting waypoints if they are specified); if no destination is specified, vehicle will randomly walk after visiting waypoints
  - SightseeingModel – vehicle randomly walks until it is a certain distance from the starting point; the vehicle then takes the shortest path back to the starting point and starts again along a different path

6.6.3.3 VISUALIZERS

Several visual displays that can exist in the main application workspace are defined. These displays are implemented using the Qt user interface specification, but the details are hidden away by the Visualizer, TableVisualizer, and other classes. Specifically, the Visualizer class manages the
display widget for the programmer, and the TableVisualizer class does the same for a QTable-based widget. To design visual displays based on other types of widgets, such as text editors, the recommended procedure is to create a new generic type of visualizer that has as a member variable the desired type of widget, binds all useful signals and slots of that widget to class member functions, and handles creation and destruction of the widget gracefully. The Visualizer and TableVisualizer classes are two good examples of such generic visualization classes. Then, one simply has to derive a specific class, such as a MapVisual or a CarListVisual, from the appropriate generic class and implement the display’s particular behavior.

## 7 SYSTEM DESIGN

### 7.1 COMPONENT DIAGRAM

The component diagram which is included above intends to show basic relationships among hardware and software components required for the project and their interfaces. The listed hardware components are GPS, ZigBee module, sensor, PC-compatible unit and
embedded computation unit, and the only listed software component is Babylon Traffic Information System which constitutes the main workload of the project.

Babylon Traffic Information System is the software that is designed to run on a UNIX-based operating system for a PC-compatible unit for vehicles including automobiles, fire fighter cars, police cars and ambulances and road side facilities including hospitals, police stations and fire houses. Yet the software needs to run on a real time operating system (RTOS) for embedded computation units for traffic signs including parking lots informers, and electronic warning devices. The software includes a Graphical User Interface as an application.

For moving agents, the system interacts with GPS at a high extent via NMEA interface. Sensors are attached to vehicles to detect accidents. They communicate with the serial port for PC-compatible unit via the sensor interface.

Zigbee Module is the configurable hardware component that enables networking. It communicates with serial ports of either PC-compatible unit or the embedded computation unit via the Zigbee interface.

7.2 PACKAGE DIAGRAM

The package diagram in Figure 7 intends to show object-oriented nature of the basic software components of the Babylon Traffic Information System.

![PACKAGE DIAGRAM](image-url)
It includes packages ZigBee Communication, GPS, Vehicles, Road Side Facilities, Traffic Signs, Map, Java Infrastructure and OpenMap API.

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

Vehicles, Road Side Facilities and Traffic Signs packages represent the main agents of the project. They all communicate with each other; hence they import the ZigBee Communication package which includes classes and methods for networking.

GPS package which includes classes and methods for global positioning of agents is imported only by Vehicles package since the latter agents are stationary.

Map package imports Vehicles, Road Side Facilities and Traffic Signs packages and includes classes and methods for depicting all agents and locations on a map via the interface. It also imports necessary classes and functions from OpenMap API.

### 7.3 ZigBee Communication Package

**FIGURE 8: CLASS DIAGRAMS FOR ZIGBEE COMMUNICATION PACKAGE**
There are four main classes for ZigBee Communication package:

- Message class includes attributes and methods to represent a message in several formats and conversion among these formats to have them arranged suitably for send and receive operations. Beyond format conversion methods listed in the diagram for this class, also makes it possible to set and retrieve attributes of the class.
- Transmitter class includes a method to send a message according to ZigBee protocols. It is in aggregation relationship with Message and Config classes since it includes objects from these classes.
- Receiver class includes methods to receive messages according to ZigBee protocols and to run threads which control whether any messages have arrived. It is in aggregation relationship with Message and Config classes since it includes objects from these classes.
- Config class includes methods to configure the ZigBee module accessed by an interface.

### 7.3.1 DATA DICTIONARY FOR THE ZIGBEE COMMUNICATION PACKAGE

#### 7.3.1.1 MESSAGE CLASS

<table>
<thead>
<tr>
<th>Attribute/method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type:int</td>
<td>Type of the message</td>
</tr>
<tr>
<td>destination:long</td>
<td>64-bit address of the destination</td>
</tr>
<tr>
<td>source:long</td>
<td>64-bit address of the source</td>
</tr>
<tr>
<td>time:java.util.Date</td>
<td>Time at which the message was generated</td>
</tr>
<tr>
<td>length:int</td>
<td>Length of the message</td>
</tr>
<tr>
<td>id:int</td>
<td>Unique message identification number</td>
</tr>
<tr>
<td>content:String</td>
<td>Content of the message</td>
</tr>
<tr>
<td>Message(content:String)</td>
<td>Constructor of the Message class, performs conversion.</td>
</tr>
</tbody>
</table>
**Detailed Design Report**

<table>
<thead>
<tr>
<th>int getType()</th>
<th>Returns the value of the type attribute for a given Message object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>long getDestination()</td>
<td>Returns the value of the destination attribute for a given Message object.</td>
</tr>
<tr>
<td>long getSource()</td>
<td>Returns the value of the source attribute for a given Message object.</td>
</tr>
<tr>
<td>Date getTime()</td>
<td>Returns the value of the time attribute for a given Message object.</td>
</tr>
<tr>
<td>int getLength()</td>
<td>Returns the value of the length attribute for a given Message object.</td>
</tr>
<tr>
<td>int getId()</td>
<td>Returns the value of the id attribute for a given Message object.</td>
</tr>
<tr>
<td>String getContent()</td>
<td>Returns the value of the content attribute for a given Message object.</td>
</tr>
</tbody>
</table>

### 7.3.1.2 Receiver Class

**Table 5: Receiver Class**

<table>
<thead>
<tr>
<th>Attribute/method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receiver()</td>
<td>Constructor of Receiver class which periodically checks the RS232 port for incoming data from ZigBee</td>
</tr>
<tr>
<td>run()</td>
<td>Checks the RS232 port for data, if data is available parses the data, sends produced MapEntity objects for display in Map.</td>
</tr>
</tbody>
</table>

### 7.3.1.3 Configuration Class

**Table 6: Configuration Class**

<table>
<thead>
<tr>
<th>Attribute/method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Config()</td>
<td>Constructor of Config class whose static methods can be used to change configuration of ZigBee hardware and query for network information.</td>
</tr>
<tr>
<td>static long getAddress()</td>
<td>Returns network address learnt from ZigBee hardware</td>
</tr>
<tr>
<td>static short getPanID()</td>
<td>Returns Pan ID learnt from ZigBee hardware</td>
</tr>
<tr>
<td>static short getHop()</td>
<td>Returns the number of maximum hops that a packet can use learnt from ZigBee hardware</td>
</tr>
<tr>
<td>static void setHop(hop:short)</td>
<td>Sets the maximum number of hops in ZigBee hardware</td>
</tr>
<tr>
<td>static void softReset()</td>
<td>Makes a software reset which resets all configuration values to default on ZigBee hardware</td>
</tr>
<tr>
<td>static void netReset()</td>
<td>Makes a network reset which causes network addresses and routes to be renewed</td>
</tr>
</tbody>
</table>
### 7.3.1.4 TRANSMITTER CLASS

<table>
<thead>
<tr>
<th>Attribute/method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>send (msg:Message)</td>
<td>Constructs data send string using msg object to be sent to ZigBee over RS232</td>
</tr>
</tbody>
</table>

### 7.4 GPS PACKAGE

There are two main classes for GPS package.

- GPSconfig class has not been elaborated yet, but it includes methods for communication with GPS device via the NMEA interface.
- GPSdata class includes attributes and methods to store and parse NMEA sentences into several fields.

GPSdata and GPSconfig classes are associated with each other as indicated by the diagram.

#### 7.4.1 DATA DICTIONARY FOR THE GPS PACKAGE

#### 7.4.1.1 GPSDATA CLASS

<table>
<thead>
<tr>
<th>Attribute/method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>longitude:int</td>
<td>The longitude of the agent with GPS device.</td>
</tr>
<tr>
<td>latitude:int</td>
<td>The latitude of the agent with GPS device.</td>
</tr>
</tbody>
</table>
### 7.5 Vehicles Package

Vehicles package comprises of nine main classes.

- Vehicle class is the base class which represents a vehicle en-route. It has attributes depicting basic properties of a vehicle and methods to access its attributes.
- Fire Fighter class is a derived class of Vehicle class and represents a fire fighter car en route.
- Automobile class is a derived class of Vehicle class and represents an automobile en route.
- Police class is a derived class of Vehicle class and represents a police car en route.
- Ambulance class is a derived class of Vehicle class and represents an ambulance en route.
- GPSwatcher class has methods to receive information from the GPS device to which the vehicle is connected. This class is an extension of a Java thread and its objects repetitively updates attributes of Vehicle objects.
- Location class represents position information of Vehicle objects.
- SensorWatcher class is an extension of a Java thread and it repetitively checks whether an emergency has occurred and then the particular Vehicle object broadcasts emergency signals in case of emergencies.
- Informer class is an extension of a Java thread and it enables communication among Vehicle objects and other agents.

<table>
<thead>
<tr>
<th>speed:double</th>
<th>The speed of the agent with GPS device.</th>
</tr>
</thead>
<tbody>
<tr>
<td>heading:double</td>
<td>The heading of the agent with GPS device.</td>
</tr>
<tr>
<td>GPSdata(sentence:String)</td>
<td>Constructor of the GPSdata class that parses a NMEA sentence.</td>
</tr>
<tr>
<td>int getLongitude()</td>
<td>Returns the value of the longitude attribute for a given GPSdata object.</td>
</tr>
<tr>
<td>int getLatitude()</td>
<td>Returns the value of the latitude attribute for a given GPSdata object.</td>
</tr>
<tr>
<td>double getSpeed()</td>
<td>Returns the value of the speed attribute for a given GPSdata object.</td>
</tr>
<tr>
<td>double getHeading()</td>
<td>Returns the value of the heading attribute for a given GPSdata object.</td>
</tr>
</tbody>
</table>
Vehicle class is in aggregation relationship with GPSWatcher, Location and SensorWatcher classes since it makes use of objects and/or methods from these classes. Each derived class of the Vehicle class is in aggregation relationship with Informer class similarly.

7.5.1 DATA DICTIONARY FOR VEHICLES PACKAGE

7.5.1.1 VEHICLE CLASS

<table>
<thead>
<tr>
<th>Attribute/method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>position:Location</td>
<td>The longitude and latitude of a Vehicle object.</td>
</tr>
<tr>
<td>heading:double</td>
<td>The heading of a Vehicle object.</td>
</tr>
<tr>
<td>speed:double</td>
<td>The speed of a Vehicle object.</td>
</tr>
</tbody>
</table>
### DETAILED DESIGN REPORT

<table>
<thead>
<tr>
<th>emergencyCode:boolean</th>
<th>The emergency code of a Vehicle object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vehicle(data:GPSdata)</td>
<td>Constructor of the Vehicle class that makes use of the GPS data</td>
</tr>
<tr>
<td>void setPosition(loc:Location)</td>
<td>Sets the value of the position attribute to a Location object for a given Vehicle object.</td>
</tr>
<tr>
<td>Location getPosition()</td>
<td>Returns the value of the position attribute for a given Vehicle object.</td>
</tr>
<tr>
<td>double getSpeed()</td>
<td>Returns the value of the speed attribute for a given Vehicle object.</td>
</tr>
<tr>
<td>double getHeading()</td>
<td>Returns the value of the heading attribute for a given Vehicle object.</td>
</tr>
<tr>
<td>void setHeading(hdg:double)</td>
<td>Sets the value of the heading attribute to a value for a given Vehicle object.</td>
</tr>
<tr>
<td>void setSpeed(spd:double)</td>
<td>Sets the value of the speed attribute to a value for a given Vehicle object.</td>
</tr>
<tr>
<td>void setEmergency(emg:boolean)</td>
<td>Sets the value of the position attribute to a logical value for a given Vehicle object.</td>
</tr>
<tr>
<td>boolean getEmergency()</td>
<td>Returns the value of the emergencyCode attribute for a given Vehicle object.</td>
</tr>
<tr>
<td>void broadcast_emergency_call()</td>
<td>Broadcasts emergency signal to other agents in case of emergencies.</td>
</tr>
</tbody>
</table>

### 7.5.1.2 FIRE FIGHTER, AMBULANCE, POLICE AND AUTOMOBILE CLASSES

TABLE 10: FIRE FIGHTER, AMBULANCE, POLICE AND AUTOMOBILE CLASSES

<table>
<thead>
<tr>
<th>Attribute/method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type:Char</td>
<td>Type of the Vehicle.</td>
</tr>
</tbody>
</table>

### 7.5.1.3 GPSWATCHER CLASS
### TABLE 11: GPSWATCHER CLASS

<table>
<thead>
<tr>
<th>Attribute/method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void run ()</td>
<td>Runs the thread at pre-defined intervals to update values of attributes of the Vehicle objects.</td>
</tr>
</tbody>
</table>

### 7.5.1.4 LOCATION CLASS

### TABLE 12: LOCATION CLASS

<table>
<thead>
<tr>
<th>Attribute/method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>longitude:int</td>
<td>The longitude of a Location object.</td>
</tr>
<tr>
<td>latitude:int</td>
<td>The latitude of a Location object.</td>
</tr>
<tr>
<td>Location(lng:int, lat:int)</td>
<td>The constructor of the Location class.</td>
</tr>
<tr>
<td>int getLongitude ()</td>
<td>Returns the value of the longitude attribute for a given Location object.</td>
</tr>
<tr>
<td>int getLatitude ()</td>
<td>Returns the value of the latitude attribute for a given Location object.</td>
</tr>
<tr>
<td>void setLatitude(lat:int)</td>
<td>Sets the value of the latitude attribute to a value for a given Location object.</td>
</tr>
<tr>
<td>void setLongitude(lng:int)</td>
<td>Sets the value of the emergencyCode attribute to a value for a given Location object.</td>
</tr>
</tbody>
</table>

### 7.5.1.5 SENSORWATCHER CLASS

### TABLE 13: SENSORWATCHER CLASS

<table>
<thead>
<tr>
<th>Attribute/method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void run ()</td>
<td>Runs the thread at pre-defined intervals to detect if an accident has occurred.</td>
</tr>
</tbody>
</table>

### 7.5.1.6 INFORMER CLASS
7.6 ROAD SIDE FACILITIES PACKAGE

Road Side Facilities package comprises of seven main classes:

- RoadSideFacility class is the base class which represents a stationary agent. It has attributes depicting basic properties of a road side facility and methods to access its attributes.
- Hospital class is a derived class of RoadSideFacility class and represents a hospital.
- Fire House class is a derived class of RoadSideFacility class and represents an firehouse.
- Police Station class is a derived class of RoadSideFacility class and represents a police station.
- EmergencyWatcher class is an extension of a Java thread and its objects repetitively
checks whether any vehicle is involved in an accident.

- Location class represents position information of RoadSideFacility objects.
- Informer class is an extension of a Java thread and it enables communication among RoadSideFacility objects and other agents.

RoadSideFacility class is in aggregation relationship with EmergencyWatcher and Location classes since it makes use of objects and/or methods from these classes. Each derived class of the RoadSideFacility class is in aggregation relationship with Informer class similarly.

### 7.6.1 DATA DICTIONARY FOR ROAD SIDE FACILITIES PACKAGE

#### 7.6.1.1 ROADSIDEFACILITY CLASS

**TABLE 15: ROADSIDEFACILITY CLASS**

<table>
<thead>
<tr>
<th>Attribute/method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>position:Location</td>
<td>The longitude and latitude of a RoadSideFacility object.</td>
</tr>
<tr>
<td>heading:double</td>
<td>The heading of a RoadSideFacility object, 0 by default.</td>
</tr>
<tr>
<td>speed:double</td>
<td>The speed of a RoadSideFacility object, 0 by default.</td>
</tr>
<tr>
<td>void setPosition(loc:Location)</td>
<td>Sets the value of the position attribute to a Location object for a given RoadSideFacility object.</td>
</tr>
<tr>
<td>Location getPosition()</td>
<td>Returns the value of the position attribute for a given RoadSideFacility object.</td>
</tr>
<tr>
<td>double getSpeed()</td>
<td>Returns the value of the speed attribute for a given RoadSideFacility object.</td>
</tr>
<tr>
<td>double getHeading()</td>
<td>Returns the value of the heading attribute for a given RoadSideFacility object.</td>
</tr>
<tr>
<td>void setHeading(hdg:double)</td>
<td>Sets the value of the heading attribute to a value for a given RoadSideFacility object.</td>
</tr>
<tr>
<td>void setSpeed(spd:double)</td>
<td>Sets the value of the speed attribute to a value for a given RoadSideFacility object.</td>
</tr>
</tbody>
</table>

#### 7.6.1.2 HOSPITAL, FIRE HOUSE, POLICE STATION CLASSES
### TABLE 16: HOSPITAL, FIRE HOUSE, POLICE STATION CLASSES

<table>
<thead>
<tr>
<th>Attribute/method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type:Char</td>
<td>Type of the RoadSideFacility.</td>
</tr>
</tbody>
</table>

---

### 7.6.1.3 EMERGENCYWATCHER CLASS

### TABLE 17: EMERGENCYWATCHER CLASS

<table>
<thead>
<tr>
<th>Attribute/method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void run ()</td>
<td>Runs the thread at pre-defined intervals to check whether any vehicle is involved in an accident.</td>
</tr>
</tbody>
</table>

---

### 7.6.1.4 LOCATION CLASS

### TABLE 18: LOCATION CLASS

<table>
<thead>
<tr>
<th>Attribute/method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>longitude:int</td>
<td>The longitude of a Location object.</td>
</tr>
<tr>
<td>latitude:int</td>
<td>The latitude of a Location object.</td>
</tr>
<tr>
<td>Location(lng:int, lat:int)</td>
<td>The constructor of the Location class.</td>
</tr>
<tr>
<td>int getLongitude ()</td>
<td>Returns the value of the longitude attribute for a given Location object.</td>
</tr>
<tr>
<td>int getLatitude ()</td>
<td>Returns the value of the latitude attribute for a given Location object.</td>
</tr>
<tr>
<td>void setLatitude(lat:int)</td>
<td>Sets the value of the latitude attribute to a value for a given Location object.</td>
</tr>
<tr>
<td>void setLongitude(lng:int)</td>
<td>Sets the value of the emergencyCode attribute to a value for a given Location object.</td>
</tr>
</tbody>
</table>

---

### 7.6.1.5 INFORMER CLASS
### Table 19: Informer Class

<table>
<thead>
<tr>
<th>Attribute/method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void run()</td>
<td>Runs the thread at pre-defined intervals to control communication with other agents.</td>
</tr>
</tbody>
</table>

## 7.7 Traffic Signs Package

Traffic Signs package comprises of three main classes.

- **TrafficSign class** represents a stationary traffic sign or a parking lot. It has attributes depicting basic properties of a traffic sign and methods to access its attributes.
- **Location class** represents position information of TrafficSign objects.
- **Informer class** is an extension of a Java thread and it enables communication among TrafficSign objects and other agents.

TrafficSign class is in aggregation relationship with Informer and Location classes since it makes use of objects and/or methods from these classes.

### 7.7.1 Data Dictionary for Traffic Signs Package
### 7.7.1.1 TRAFFICSIGN CLASS

**TABLE 20: TRAFFICSIGN CLASS**

<table>
<thead>
<tr>
<th>Attribute/method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>position:Location</td>
<td>The longitude and latitude of a TrafficSign object.</td>
</tr>
<tr>
<td>heading:double</td>
<td>The heading of a TrafficSign object, 0 by default.</td>
</tr>
<tr>
<td>speed:double</td>
<td>The speed of a TrafficSign object, 0 by default.</td>
</tr>
<tr>
<td>void setPosition(loc:Location)</td>
<td>Sets the value of the position attribute to a Location object for a given TrafficSign object.</td>
</tr>
<tr>
<td>Location getPosition( )</td>
<td>Returns the value of the position attribute for a given TrafficSign object.</td>
</tr>
<tr>
<td>double getSpeed( )</td>
<td>Returns the value of the speed attribute for a given TrafficSign object.</td>
</tr>
<tr>
<td>double getHeading( )</td>
<td>Returns the value of the heading attribute for a given TrafficSign object.</td>
</tr>
<tr>
<td>void setHeading(hdg:double)</td>
<td>Sets the value of the heading attribute to a value for a given TrafficSign object.</td>
</tr>
<tr>
<td>void setSpeed(spd:double)</td>
<td>Sets the value of the speed attribute to a value for a given TrafficSign object.</td>
</tr>
</tbody>
</table>

### 7.7.1.2 LOCATION CLASS

**TABLE 21: LOCATION CLASS**

<table>
<thead>
<tr>
<th>Attribute/method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>longitude:int</td>
<td>The longitude of a Location object.</td>
</tr>
<tr>
<td>latitude:int</td>
<td>The latitude of a Location object.</td>
</tr>
<tr>
<td>Location(lng:int, lat:int)</td>
<td>The constructor of the Location class.</td>
</tr>
<tr>
<td>int getLongitude ( )</td>
<td>Returns the value of the longitude attribute for a given Location object.</td>
</tr>
<tr>
<td>int getLatitude ( )</td>
<td>Returns the value of the latitude attribute for a given Location object.</td>
</tr>
</tbody>
</table>
void setLatitude(lat:int)  
Sets the value of the latitude attribute to a value for a given Location object.

void setLongitude(lng:int)  
Sets the value of the emergencyCode attribute to a value for a given Location object.

### 7.7.1.3 INFORMER CLASS

<table>
<thead>
<tr>
<th>Attribute/method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void run ()</td>
<td>Runs the thread at pre-defined intervals to control communication with other agents.</td>
</tr>
</tbody>
</table>

### 7.8 MAP PACKAGE

Map package comprises of four main classes and an interface:

- Map class represents required objects and methods for drawing and finding shortest
paths between two destinations.

- Location class represents position information of objects included in the map.
- MapRefresher class is an extension of a Java thread and it refreshes the map at pre-defined intervals.
- MapEntity represents all the agent-objects residing on the map of a given location.

Map class is in aggregation relationship with MapEntity class since it includes MapEntity objects. Similarly, MapEntity class is in aggregation relationship with Location class, MapRefresher class is in aggregation with Map class, the GUI is in aggregation with Map class.

### 7.8.1 DATA DICTIONARY FOR MAP PACKAGE

#### 7.8.1.1 MAP CLASS

<table>
<thead>
<tr>
<th>Attribute/method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>position:ArrayList&lt;MapEntity&gt;</td>
<td>A list of MapEntity objects.</td>
</tr>
<tr>
<td>void redraw( )</td>
<td>Draw function of a Map object.</td>
</tr>
<tr>
<td>ArrayList&lt;Location&gt; sPath (loc:Location, dest:Location)</td>
<td>Finds shortest path between two destinations.</td>
</tr>
</tbody>
</table>

#### 7.8.1.2 MAPENTITY CLASS

<table>
<thead>
<tr>
<th>Attribute/method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>position:Location</td>
<td>The longitude and latitude of a MapEntity object.</td>
</tr>
<tr>
<td>heading:double</td>
<td>The heading of a MapEntity object.</td>
</tr>
<tr>
<td>speed:double</td>
<td>The speed of a MapEntity object.</td>
</tr>
<tr>
<td>emergencyCode:boolean</td>
<td>The emergency code of a MapEntity object.</td>
</tr>
<tr>
<td>MapEntity(pos:Location,spd:double,hdg:double,emg:boolean)</td>
<td>The constructor of the MapEntity class that maps each agent to a MapEntity object with construction.</td>
</tr>
</tbody>
</table>
### 7.8.1.3 LOCATION CLASS

**TABLE 25: LOCATION CLASS**

<table>
<thead>
<tr>
<th>Attribute/method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>longitude:int</td>
<td>The longitude of a Location object.</td>
</tr>
<tr>
<td>latitude:int</td>
<td>The latitude of a Location object.</td>
</tr>
<tr>
<td>Location(lng:int, lat:int)</td>
<td>The constructor of the Location class.</td>
</tr>
<tr>
<td>int getLongitude( )</td>
<td>Returns the value of the longitude attribute for a given Location object.</td>
</tr>
<tr>
<td>int getLatitude( )</td>
<td>Returns the value of the latitude attribute for a given Location object.</td>
</tr>
<tr>
<td>void setLatitude(lat:int)</td>
<td>Sets the value of the latitude attribute to a value for a given Location object.</td>
</tr>
<tr>
<td>void setLongitude(lng:int)</td>
<td>Sets the value of the emergencyCode attribute to a value for a given Location object.</td>
</tr>
</tbody>
</table>
TABLE 26: MAPREFRESHER CLASS

<table>
<thead>
<tr>
<th>Attribute/method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interval:int</td>
<td>The interval at which a map is refreshed.</td>
</tr>
<tr>
<td>void run ()</td>
<td>Runs the thread at pre-defined intervals to refresh map at certain intervals.</td>
</tr>
<tr>
<td>MapRefresher(intv:int)</td>
<td>The constructor of a MapRefresher object.</td>
</tr>
</tbody>
</table>

TABLE 27: GUI INTERFACE

<table>
<thead>
<tr>
<th>Attribute/method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapp:Map</td>
<td>The map included in the interface.</td>
</tr>
<tr>
<td>menu:java.swing</td>
<td>An empty menu object.</td>
</tr>
<tr>
<td>void processClick(x:int,y:int)</td>
<td>The function that controls mouse movements in the interface.</td>
</tr>
</tbody>
</table>

7.8.1.5 GUI INTERFACE

7.9 SEQUENCE DIAGRAMS

The sequence diagram depicting an instance of the system when no vehicle is involved in accidents is included above. This diagram intends to show data flow between inter-system agents and the end user. In the diagram the end user is depicted as an automobile driver.

An execution of the system begins with the user's initialization of the graphical user interface. As he/she moves in his/her automobile, the transmitter placed in the vehicle sends location, speed and heading information of the vehicle to other automobiles, road side facilities and traffic signs.

Simultaneously, via the receiver placed in the vehicle, location, speed and heading information and if exists warning messages are received from other agents. The driver sees this piece of information on the graphical user interface if he/she would like to.

The second sequence diagram showing an instance of the system when the driver's automobile is involved in an accident. In this case, the sensors attached in the car sense the
accident and pass this value to the software and the vehicle broadcasts an emergency signal via its transmitter.

Driver's automobile sends emergency signals to other automobiles, other vehicles, traffic signs other than parking lots, and road side facilities. Other automobiles see the emergency signal generated by the car in accident, yet they do not respond to this signal.

In case of the accident, the graphical user interface of the automobile does not show other vehicles around if it is not affected by the accident. Yet, nearby firefighter cars, police cars and ambulances respond to the emergency signal by sending their location, speed and heading information together with their availability about the accident. These types of vehicles are depicted in the graphical user interface.

Road side facilities respond to the car by sending their location and response to the accident simultaneously. They also send warning messages to nearby ambulances, fire fighter cars and ambulances. Traffic signs other than parking lots receive the accident information and broadcast warning messages to other vehicles and agents nearby. Hence the driver who is involved in the accident sees also information about his/her accident and the road side facilities' information and response to the accident if the graphical user interface can work after crash.
AN INSTANCE OF THE SYSTEM

[NO VEHICLE IS INVOLVED IN AN ACCIDENT]

FIGURE 14: SEQUENCE DIAGRAM IN CASE OF NO ACCIDENT
FIGURE 15: SEQUENCE DIAGRAM IN CASE OF AN ACCIDENT

AN INSTANCE OF THE SYSTEM
[ DRIVER'S VEHICLE IS INVOLVED IN AN ACCIDENT ]

Automobile  Automobile  Firefighter  Police  Ambulance  Traffic Sign  Hospital  Parking Lot  Police Station  Firehouse

Driver

- sends emergency signal
- sends the location and the response of the hospital
- sends emergency signal
- sends the location and the response of the police station
- sends the location and the response of the firehouse
- sends the location and the content of the warning of the traffic sign
- sends the location and the response of the hospital
- sends emergency signal
- sends the location and the content of the warning of the traffic sign
- sends the location and the response of the firehouse
- sends emergency signal
- sends the location and the content of the warning of the traffic sign
- sends the location and the response of the police station
- sends emergency signal
- sends the location and the content of the warning of the traffic sign
- sends the location and the response of the firehouse
- sends emergency signal

In case of an accident:
- sends location and vehicle ID to the police
- sends location and vehicle ID to the ambulance
- sends location and vehicle ID to the hospital
- sends location and vehicle ID to the traffic sign
- sends location and vehicle ID to the firehouse
FIGURE 16: SEQUENCE DIAGRAM FOR SETTING A ZOOM LEVEL VIA USER INTERFACE

FIGURE 17: SEQUENCE DIAGRAM FOR FINDING SHORTEST PATH BETWEEN CURRENT LOCATION AND DESTINATION
7.10 STATE AND ACTIVITY DIAGRAM

FIGURE 18: STATE DIAGRAM
The State diagram in Figure 17 and the activity diagram in Figure 18 intend to depict general execution sequences in the system. User can receive information from the map interface. The system execution begins with turning-on of the UI. User can decide to see the information of his/her vehicle on the interface at any time.

Similarly the user can make decisions of seeing other vehicles' information and receiving warning messages from road side equipments, timelessly. In case of emergencies involving the user's vehicle, broadcasting the emergency signal dominates the program execution. Unless emergencies occur, the system continues its normal execution mentioned above.

User can navigate to another location and find the shortest route between two destinations at any time by using the interface.

User can terminate the system by turning off the UI.

FIGURE 19: ACTIVITY DIAGRAM
8 SCHEDULE

8.1 TASKS ACCOMPLISHED

This section provides a summary of tasks accomplished so far.

- Information collected on the following topics,
  - Mapping APIs: Google Map API, OpenStreetMap API, Gmap Catcher
  - Simulator tools: NS-2, GrooveNet, SUMO
  - Development tools: Java2D API, Java development tools
  - ZigBee wireless communication technology specifications

- GUI design

- Requirements Specifications are defined
  - Functional/Nonfunctional requirements
  - Use Cases and corresponding diagrams

- Architectural Design
  - Class diagramming
  - Component diagramming
  - Sequence diagramming
  - Data Dictionaries

- Literature Survey
  - FleetNet project
  - VITP (Vehicular information transfer protocol)
  - Lochert C., NEC Europe LTD., Hartenstein H., NEC Europe LTD., Tian J., Univ. of Stuttgart, Füssler H., Univ. of Mannheim, Hermann D., DaimlerChrysler AG, Mauve M., Univ. of Mannheim, “A Routing Strategy for Vehicular Ad Hoc Networks in City Environments”
  - Dikaikos D.M., Dept. of CS, Univ. of Cyprus, Florides A., Dept. of CS, Univ. of Cyprus, Nadeem T., Siemens Corporate Research, Ifode L., Dept. of CS, Univ. of Rutgers, “Location-aware services over vehicular ad-hoc networks using car to car communication”
8.2 DIVISION OF LABOR

Vehicular Interface: Tugberk, Okan, Arif

Network: Anıl, Okan

GUI: Anıl, Arif

Simulators: Anıl, Tugberk

Map: Tugberk, Arif

8.3 GANTT CHART

<table>
<thead>
<tr>
<th>ID</th>
<th>Task Description</th>
<th>Start</th>
<th>Finish</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Graphical User Interface Design and Implementation</td>
<td>22/02/2010</td>
<td>19/03/2010</td>
<td>4w</td>
</tr>
<tr>
<td>2</td>
<td>Implementation of main window</td>
<td>22/02/2010</td>
<td>26/02/2010</td>
<td>1w</td>
</tr>
<tr>
<td>3</td>
<td>Implementation of menu and configuration windows</td>
<td>01/03/2010</td>
<td>19/03/2010</td>
<td>3w</td>
</tr>
<tr>
<td>4</td>
<td>Map functions implementation</td>
<td>01/03/2010</td>
<td>09/04/2010</td>
<td>6w</td>
</tr>
<tr>
<td>5</td>
<td>Implementation of classes in map package</td>
<td>01/03/2010</td>
<td>19/03/2010</td>
<td>3w</td>
</tr>
<tr>
<td>6</td>
<td>Integration of OpenMap API</td>
<td>09/03/2010</td>
<td>19/03/2010</td>
<td>2w</td>
</tr>
<tr>
<td>7</td>
<td>Implementation of entity representation</td>
<td>09/03/2010</td>
<td>28/03/2010</td>
<td>3w</td>
</tr>
<tr>
<td>8</td>
<td>Implementation of path finding functionality</td>
<td>15/03/2010</td>
<td>09/04/2010</td>
<td>4w</td>
</tr>
<tr>
<td>9</td>
<td>Testing and debugging</td>
<td>05/04/2010</td>
<td>16/04/2010</td>
<td>2w</td>
</tr>
<tr>
<td>10</td>
<td>Core system implementation</td>
<td>22/02/2010</td>
<td>02/04/2010</td>
<td>6w</td>
</tr>
<tr>
<td>11</td>
<td>Implementation of Vehicle classes</td>
<td>22/02/2010</td>
<td>05/03/2010</td>
<td>2w</td>
</tr>
<tr>
<td>12</td>
<td>Implementation of GPS classes</td>
<td>01/03/2010</td>
<td>19/03/2010</td>
<td>3w</td>
</tr>
<tr>
<td>13</td>
<td>Implementation of traffic sign classes</td>
<td>15/03/2010</td>
<td>28/03/2010</td>
<td>2w</td>
</tr>
<tr>
<td>14</td>
<td>Implementation of Roadside Facility classes</td>
<td>15/03/2010</td>
<td>02/04/2010</td>
<td>3w</td>
</tr>
<tr>
<td>15</td>
<td>Modifying Simulator source code</td>
<td>05/04/2010</td>
<td>16/04/2010</td>
<td>2w</td>
</tr>
<tr>
<td>16</td>
<td>Integrating Simulator with the system</td>
<td>12/04/2010</td>
<td>30/04/2010</td>
<td>3w</td>
</tr>
<tr>
<td>17</td>
<td>Network communication implementation</td>
<td>12/04/2010</td>
<td>30/04/2010</td>
<td>3w</td>
</tr>
<tr>
<td>18</td>
<td>Integration of system modules</td>
<td>01/03/2010</td>
<td>23/04/2010</td>
<td>8w</td>
</tr>
<tr>
<td>19</td>
<td>Testing and debugging</td>
<td>03/05/2010</td>
<td>14/05/2010</td>
<td>2w</td>
</tr>
<tr>
<td>20</td>
<td>User documentation</td>
<td>03/05/2010</td>
<td>14/05/2010</td>
<td>2w</td>
</tr>
<tr>
<td>21</td>
<td>Packaging and Release</td>
<td>15/05/2010</td>
<td>28/05/2010</td>
<td>3w</td>
</tr>
</tbody>
</table>

FIGURE 20: GANTT CHART
9 BIBLIOGRAPHY


3. Dikaikos, Iqbal, Nadeem and Iftode. Location aware services over vehicular ad hoc system using car to car communication.


