Group Name: Babylon

Group Members:
- M. Tuğberk İşyapar
- Arif S. Özçelik
- Okan T. Tursun
- Ö. Anıl Türkkan

18, December 2009
# CONTENTS

INTRODUCTION..........................................................................................................................5  
Purpose of Document ..................................................................................................................5  
Problem Definition ....................................................................................................................5  
Project Goals ..............................................................................................................................5  
Changes in Functionalities after delivering SRS ........................................................................6  

DESIGN CONSTRAINTS...............................................................................................................6  
Time Constraints .......................................................................................................................6  
Resource Constraints ...............................................................................................................6  
Performance Constraints .........................................................................................................6  
Software Constraints ................................................................................................................7  
Hardware Constraints ...............................................................................................................7  

MODELING OF THE SYSTEM....................................................................................................7  
Overview of System ..................................................................................................................7  
Vehicles .....................................................................................................................................8  
Roadside Equipment .................................................................................................................8  
Coordinators .............................................................................................................................8  
GUI Design ...............................................................................................................................9  
Menu Bar ..................................................................................................................................9  
Controls ..................................................................................................................................10  
Map .........................................................................................................................................10  
Network Message Format .........................................................................................................11  
Performance of Network ..........................................................................................................11  

TECHNOLOGIES USED IN PROJECT......................................................................................12  
Network Structure ...................................................................................................................12  
Possible Issues Regarding Implementation ............................................................................14  
GPS NMEA ...............................................................................................................................14  
Data link layer ..........................................................................................................................15  
General Sentence Format .........................................................................................................15  
JAVA Libraries .........................................................................................................................16  
JAVA2D API .............................................................................................................................16  
JAVA SWING API ....................................................................................................................16  
OpenStreetMap ( OSM ) ..........................................................................................................17  
Simulators ................................................................................................................................18  
GrooveNet (7) .........................................................................................................................18  
TraNS (8) ..................................................................................................................................19  
SUMO (9) ..................................................................................................................................20
SYSTEM DESIGN ........................................................................................................................................21
Component Diagram ..................................................................................................................................21
Package Diagram .......................................................................................................................................22
Zigbee Communication Package ..................................................................................................................23
  Data Dictionary for Zigbee Communication Package ..............................................................................24
GPS Package ...............................................................................................................................................26
  Data Dictionary for GPS Package ...........................................................................................................26
Vehicles Package .........................................................................................................................................27
  Data Dictionary for Vehicles Package ...................................................................................................29
Road Side Facilities Package .......................................................................................................................31
  Data Dictionary for Road Side Facilities Package ..................................................................................32
Traffic Signs Package ..................................................................................................................................34
  Data Dictionary for Traffic Signs Package .............................................................................................35
MAP PACKAGE ..........................................................................................................................................36
  Data Dictionary for Traffic Signs Package .............................................................................................37
SEQUENCE DIAGRAMS .............................................................................................................................39
USE CASES ................................................................................................................................................43
SCHEDULE ................................................................................................................................................46
  Tasks Accomplished ...............................................................................................................................46
Division Of Labor ........................................................................................................................................47
Gantt Chart ................................................................................................................................................47
Bibliography ................................................................................................................................................48

Table 1: VEHICULAR Information Transfer Protocol (VITP) .................................................................11
Table 2: XBEE PRO Znet 2.5 Technical Specifications ........................................................................13
Table 3: Data LINK LAYER ......................................................................................................................15
Table 4: Zigbee Message Class ..................................................................................................................24
Table 5: ZigBee Receiver Class ................................................................................................................25
Table 6: Zigbee Config Class ....................................................................................................................25
Table 7: Zigbee Transmitter Class ...............................................................................................................26
Table 8: GPSdata Class ............................................................................................................................27
Table 9: VEHICLE CLASS .........................................................................................................................29
Table 10: FIRE FIGHTER, AMBULANCE, POLICE AND AUTOMOBILE CLASSES ..................30
Table 11: GPSWATCHER CLASS ...........................................................................................................30
Table 12: LOCATION CLASS ....................................................................................................................30
Table 13: SensorWatcher Class ................................................................................................................31
Table 14: INFORMER CLASS ....................................................................................................................31
Table 15: ROADSIDE FACILITY CLASS ................................................................................................32
Table 16: HOSPITAL, FIRE HOUSE, POLICE STATION CLASSES ..................................................33
Table 17: EmergencyWatcher Class ........................................................................................................33
Table 18: Location Class ...........................................................................................................................33
Table 19: Informer Class ............................................................................................................................34
INITIAL DESIGN REPORT

Table 20: TrafficSign Class

Table 21: Location Class

Table 22: Informer Class

Table 23: Map Class

Table 24: MapEntity Class

Table 25: Location Class

Table 26: MapRefresher Class

Table 27: GUI Interface

Figure 1: Roles of nodes in the vehicular ad hoc network

Figure 2: BABYLON Vehicular Ad-Hoc Network GUI design

Figure 3: 1 XBee Pro Znet 2.5 transceiver hardware

Figure 4: Component Diagram

Figure 5: Package Diagram

Figure 6: CLASS Diagrams for ZigBee Communication Package

Figure 7: Class Diagrams for GPS Package

Figure 8: Class Diagrams for Vehicles Package

Figure 9: Class Diagrams for Road Side Facilities Package

Figure 10: Class Diagrams for Traffic Signs Package

Figure 11: Class Diagrams for Map Package

Figure 12: SEQUENCE DIAGRAM IN CASE OF NO ACCIDENT PRESENT

Figure 13: SEQUENCE DIAGRAM IN CASE OF AN accident

Figure 14: Use Case Diagram for Driver

Figure 15: Use Case Diagram for Vehicles

Figure 16: Use Case Diagram for Road Side Facilities

Figure 17: Use Case Diagram for Traffic Signs

Figure 18: Gantt Chart
INTRODUCTION

PURPOSE OF 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.

PROBLEM DEFINITION

Traffic and transportation systems on roads get more and more complex everyday. 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. Our traffic system lacks a tool of the kind defined.

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.
CHANGES IN FUNCTIONALITIES AFTER DELIVERING SRS

At first, the user interface was thought to have two alternating modes.

- On one mode there was going to be radar displaying vehicles and other necessary information like traffic density, emergency signals, hospitals, police stations etc.
- On the second mode there was going to be a map showing the driver roads and his/her vehicles location on map. The map was also going to be used for finding and optimizing a path between two locations.

These two modes will be combined and there will be only a single user interface for all actions.

- There will be a zooming functionality of this interface.
- There may be a coordinator, listing GPS data of all the vehicles and road equipment in its range.

DESIGN CONSTRAINTS

TIME CONSTRAINTS

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

RESOURCE CONSTRAINTS

Although the Istanbul Metropolitan Municipality 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. As we may have a chance of finding guidance through resources, even if found, the help they provide would probably be very limited.

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.

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 OpenStreetMap will be used for drawing a map on the user interface.
- For drawing different vehicles on the map according to their GPS data, Java 2D classes will be considered

HARDWARE CONSTRAINTS

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

MODELING OF THE SYSTEM

OVERVIEW OF 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]
INITIAL DESIGN REPORT

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.

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.

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.

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.
GUI DESIGN

This section illustrates basics of GUI that will be implemented in Babylon 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. The GUI will consist of three main parts: Menu Bar, Controls and Map.

**FIGURE 2: BABYLON VEHICULAR AD-HOC NETWORK GUI DESIGN**

**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.
**INITIAL DESIGN REPORT**

**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.

---

**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.

---

**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.
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.

**TABLE 1: VEHICULAR INFORMATION TRANSFER PROTOCOL (VITP)**

<table>
<thead>
<tr>
<th>VITP message syntax (1)</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 Line 1 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 can be passed in message body.

**PERFORMANCE OF 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 packet 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 (2005) in “VITP: An...”
Information Transfer Protocol for Vehicular Computing”. But all these results are obtained on a network based on IEEE 802.11.

TECHNOLOGIES USED IN PROJECT

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: 1 XBEE PRO ZNET 2.5 TRANSCEIVER HARDWARE
XBeeproZnet 2.5 has the following specifications:

**TABLE 2: XBEE PRO ZNET 2.5 TECHNICAL SPECIFICATIONS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Rate</td>
<td>250 kbps</td>
</tr>
<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. 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.
INITIAL DESIGN REPORT

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.

POSSIBLE ISSUES REGARDING 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.

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 (2). The standard uses a simple
INITIAL DESIGN REPORT

ASCII, serial communications protocol that defines how data is transmitted in a "sentence" from one "talker" to multiple "listeners" at a time.

<table>
<thead>
<tr>
<th>DATA LINK LAYER</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TABLE 3: DATA LINK LAYER</strong></td>
</tr>
<tr>
<td><strong>Baud rate</strong></td>
</tr>
<tr>
<td><strong>Data bits</strong></td>
</tr>
<tr>
<td><strong>Parity</strong></td>
</tr>
<tr>
<td><strong>Stop bits</strong></td>
</tr>
<tr>
<td><strong>Handshake</strong></td>
</tr>
</tbody>
</table>

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 (3).

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

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.

QUERY SENTENCES

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

```
$tllQ,sss,<CR><LF>
```

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: $\text{SGPGQ, GGA<CR><LF>}$

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.

**GPsd**

gpdsd 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 gpdsd, multiple location-aware client applications (such as navigational and war driving software) can share access to receivers without contention or loss of data. Also, gpdsd responds to queries with a format that is substantially easier to parse than the NMEA 0183 emitted by most GPSes. The gpdsd distribution includes a linkable C service library, a C++ wrapper class, and a Python module that developers of gpdsd-aware applications can use to encapsulate all communication with gpdsd. (4)

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.

**Java Libraries**

**Java 2D API**

Java 2D is an API for drawing two-dimensional graphics using the Java programming language. Every Java 2D drawing operation can ultimately be treated as filling a shape using a paint and compositing the result onto the screen.

**Drawing Process**

A crude representation of the drawing process in Java 2D can be represented in 4 steps: (5)

1. Cast Graphics object to Graphics 2D
   
   ```java
   Graphics g; Graphics2D g2d = (Graphics2D) g;
   ```

2. Set pen parameters
   
   ```java
   g2d.setPaint(fillColorOrPattern);
   g2d.translate(...);
   g2d.rotate(...);
   g2d.scale(...);
   ```

3. Create a Shape Object
   
   ```java
   Rectangle2D.Double rect = ...;
   ```

4. Draw an outlined or filled version of the Shape
   
   ```java
   g2d.draw(someShape);
   g2d.fill(someShape);
   ```

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

**FOUNDATIONS (6)**

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.

**OPENSTREETMAP (OSM)**

OSM is a project aiming to create a free editable map of the world. It is continually under development as new data come from voluntary participants around the world. These participants, by the help of GPS devices, aerial photography, other free sources or simply local knowledge, gather information about streets, roads, buildings etc that surround them.

OSM is currently licensed under Creative Commons Attribution – ShareAlike 2.0 license. There are discussions about re-licensing it under Open Database License from Open Data Commons. In general, the Creative Commons License requires the following rules to be applied:

- **Attribution**: Licensees to copy, distribute, display and perform the work and make derivative works based on it only if they give the author or licensor the credits in the manner specified by these, or
• **Non-Commercial:** Licensees may copy, distribute, display, and perform the work and make derivative works based on it only for non-commercial purposes, or

• **No Derivative Work:** Licensees may copy, distribute, display and perform only verbatim copies of the work, not derivative works based on it, or

• **Share-Alike:** Licensees may distribute derivative works only under a license identical to the license that governs the original work.

The OSM foundation however, only requires the first and the fourth of these requirements to be satisfied.

OSM has an API for fetching and saving raw geo-data from/to the OSM database. The API is based on the ideas of the REST (Representational State Transfer) API. REST-style architectures consist of clients and servers. Clients initiate requests to servers; servers process requests and return appropriate responses. The REST requests take the form of HTTP GET, PUT, POST, and DELETE messages. Any payload is in XML form, using the MIME type "text/xml" and UTF-8 character encoding.

There is a lot of software that renders and present maps, collect map data, edit map data. A number of them for each purpose can be found on the wiki page for OSM.

**SIMULATORS**

Since the coding is not actively started, the choice for the simulator is not clear. But the simulator will be selected from the following three alternatives.

**GROOVENET (7)**

GrooveNet is developed by Professor Rahul Mangharam and Professor Raj Rajkumar of Carnegie Mellon University. GrooveNet is a hybrid simulator which enables communication between simulated vehicles, real vehicles and between real and simulated vehicles. By modeling inter-vehicular communication within a real street map-based topography it facilitates protocol design and also in-vehicle deployment. GrooveNet's modular architecture incorporates mobility, trip and message broadcast models over a variety of link and physical layer communication models. GrooveNet is coded using C++.
FEATURES

1. GrooveNet is a modular event-based simulator with well-defined model interfaces that make adding models easy. Models may be added without concern of conflicts with existing models as dependencies are resolved automatically.

2. GrooveNet supports multiple vehicle, trip and mobility models over a variety of network link and physical layer models. In order to correctly represent vehicle interaction, GrooveNet includes simple car-following, traffic lights, lane changing and simulated GPS models.

3. The graphical interface makes it easy to auto-generate simulations consisting of thousands of vehicles across any location in the US. All vehicles obey street speed limits and are displayed on the street map based on their current GPS coordinates.

4. 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.

5. 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.

6. 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.

USABILITY

GrooveNet supplies us excellent pre-defined functions that will be used during the simulation, but to fully adapt our system major changes must be done to the GrooveNet. The problem with the modification is that the project is not open source and it is forbidden to use the code without contacting the professors. A contact is tried to be established by sending e-mails in the pursuit of permission to modify the program.

TRANS (8)

TraNS (Traffic and Network Simulation Environment) is a GUI tool that integrates traffic and network simulators (SUMO and ns2) to generate realistic simulations of Vehicular Ad
hoc NETworks (VANETs). TraNS allows the information exchanged in a VANET to influence the vehicle behavior in the mobility model. For example, when a vehicle broadcasts information reporting an accident, some of the neighboring vehicles may slow down. TraNS is created in the Ecole Polytechnique Federale de Lausanne University. At the time being, development of TraNS is suspended. TraNS is an open source project. Language is JAVA.

**FEATURES**

1. Mobility trace generation for ns2 from TIGER and Shapefile maps (using the netconvert tool from SUMO)
2. Map cropping and speed rescaling (only for TIGER maps)
3. Generation of random and flow-based vehicle routes (using the duarouter tool tool from SUMO)
4. Google Earth visualization of simulations (only for TIGER maps)
5. SUMO and ns2 coupling through the TraCI interface (the traci client included in the package works only with SUMO version r5499, available on the svn)
6. Possibility to simulate road traffic events, e.g., accidents (both vehicle-related and location-related)
7. Framework for developing new VANET applications
8. Two ready-to-use VANET applications: Road Danger Warning (safety) and Dynamic Reroute (traffic efficiency)
9. Large scale simulations (tested up to 3000 vehicles)

**USABILITY**

As TraNS is an open source project, modifications to the code is not prohibited. But TraNS is somewhat inferior compared to GrooveNET when compared in existing functionality and ease of extensibility. Since it depends on SUMO and NS2 packages, using latest release of these packages will probably cause problems. However, if GrooveNET is not available for modification, TraNS will most likely to be the choice for the simulator.

**SUMO (9)**

"Simulation of Urban Mobility" (SUMO) is an open source, highly portable, microscopic road traffic simulation package designed to handle large road networks. It is mainly developed by employees of the Institute of Transportation Systems at the German Aerospace Center. SUMO is also an open source project. It contains codes and visuals to simulate road traffic. It does not contain any protocols that promote vehicular interaction. Language is also JAVA.
USABILITY

If SUMO is chosen as the simulator, then a network protocol must be added to simulate vehicular interaction. Although modifying SUMO is the most cumbersome of the three simulators, it will result in a simulator that is utterly unique to our project.

SYSTEM DESIGN

COMPONENT DIAGRAM

The component diagram which is included above intends to show basic relationships among hardware and software components required for the project. 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.

**PACKAGE DIAGRAM**

The package diagram included above intends to show object-oriented nature of the basic software components of the Babylon Traffic Information System. It includes packages Zigbee

---

**FIGURE 5: PACKAGE DIAGRAM**
INITIAL DESIGN REPORT

Communication, GPS, Vehicles, Road Side Facilities, Traffic Signs, Map and Java Infrastructure.

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.

ZIGBEE COMMUNICATION PACKAGE

FIGURE 6: 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.

### DATA DICTIONARY FOR ZIGBEE COMMUNICATION PACKAGE

#### 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>
<tr>
<td>Attribute/method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>int getType()</td>
<td>Returns the value of the type attribute for a given Message object.</td>
</tr>
<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>

### RECEIVER CLASS

**TABLE 5: ZIGBEE 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>

### CONFIG CLASS

**TABLE 6: ZIGBEE CONFIG 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 it 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</td>
</tr>
</tbody>
</table>
on ZigBee hardware

| static void netReset () | Makes a network reset which causes network addresses and routes to be renewed |

TRANSMITTER CLASS

**TABLE 7: ZIGBEE 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>

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.

DATA DICTIONARY FOR GPS PACKAGE
TABLE 8: 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>
<tr>
<td>speed:double</td>
<td>The speed of the agent with GPS device.</td>
</tr>
<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>

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.

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.

**FIGURE 8: CLASS DIAGRAMS FOR VEHICLES PACKAGE**
## DATA DICTIONARY FOR VEHICLES PACKAGE

### VEHICLE CLASS

#### TABLE 9: 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>
<tr>
<td>emergencyCode:boolean</td>
<td>The emergency code of a Vehicle object.</td>
</tr>
<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>
### INITIAL DESIGN REPORT

#### 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>

#### 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>

#### 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>

#### SENSORWATCHER CLASS
### INITIAL DESIGN REPORT

**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>

**TABLE 14: 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>

### ROAD SIDE FACILITIES PACKAGE

**FIGURE 9: CLASS DIAGRAMS FOR ROAD SIDE FACILITIES PACKAGE**

BABYLON VEHICULAR AD-HOC NETWORK
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 a 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.

### DATA DICTIONARY FOR ROAD SIDE FACILITIES PACKAGE

#### 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>
</tbody>
</table>
### INITIAL DESIGN REPORT

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>

### 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>

### 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>

### 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>
</tbody>
</table>
**INITIAL DESIGN REPORT**

<table>
<thead>
<tr>
<th>int getLatitude ( )</th>
<th>Returns the value of the latitude attribute for a given Location object.</th>
</tr>
</thead>
<tbody>
<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>

**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>

**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.

DATA DICTIONARY FOR TRAFFIC SIGNS PACKAGE

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>

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>
</tbody>
</table>

BABYLON VEHICULAR AD-HOC NETWORK
latitude:int

The latitude of a Location object.

Location(lng:int, lat:int)

The constructor of the Location class.

int getLongitude ()

Returns the value of the longitude attribute for a given Location object.

int getLatitude ()

Returns the value of the latitude attribute for a given Location object.

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.

INFORMER CLASS

**TABLE 22: 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>

MAP PACKAGE

**FIGURE 11: CLASS DIAGRAMS FOR 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.

DATA DICTIONARY FOR TRAFFIC SIGNS PACKAGE

### MAP CLASS

**TABLE 23: 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>

### MAPENTITY CLASS

**TABLE 24: 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>
</tbody>
</table>
### Emergency Code

- **emergencyCode**: boolean
  - The emergency code of a MapEntity object.

### MapEntity

- **MapEntity(pos: Location, spd: double, hdg: double, emg: boolean)**
  - The constructor of the MapEntity class that maps each agent to a MapEntity object with construction.

### Position Methods

- **void setPosition(loc: Location)**
  - Sets the value of the position attribute to a Location object for a given MapEntity object.

- **Location getPosition()**
  - Returns the value of the position attribute for a given MapEntity object.

- **double getSpeed()**
  - Returns the value of the speed attribute for a given MapEntity object.

- **double getHeading()**
  - Returns the value of the heading attribute for a given MapEntity object.

- **void setHeading(hdg: double)**
  - Sets the value of the heading attribute to a value for a given MapEntity object.

- **void setSpeed(spd: double)**
  - Sets the value of the speed attribute to a value for a given MapEntity object.

- **void setEmergency(emg: boolean)**
  - Sets the value of the emergencyCode attribute to a value for a given MapEntity object.

- **boolean getEmergency()**
  - Returns the value of the emergencyCode attribute for a given MapEntity object.

### 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>
</tbody>
</table>
void setLongitude(lng:int) | Sets the value of the emergencyCode attribute to a value for a given Location object.

**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>

**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>

**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 type 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.
FIGURE 12: SEQUENCE DIAGRAM IN CASE OF NO ACCIDENT PRESENT

AN INSTANCE OF THE SYSTEM
[NO VEHICLE IS INVOLVED IN AN ACCIDENT]

INITIAL DESIGN REPORT

BABYLON VEHICULAR AD-HOC NETWORK
FIGURE 13: SEQUENCE DIAGRAM IN CASE OF AN ACCIDENT
FIGURE 14: USE CASE DIAGRAM FOR DRIVER
FIGURE 15: USE CASE DIAGRAM FOR VEHICLES

- In case of emergencies, send signals to police stations and cars, fire houses and firefighter cars and hospitals and ambulances.
- Send signals identifying type, location, speed and heading.
- Receive/Ignore signals from other vehicles identifying their type, location, speed and heading.
- Receive/Ignore warning messages from road side facilities and traffic signs and vehicles which are involved in accidents, or police cars, fire fighter cars and ambulances.
FIGURE 16: USE CASE DIAGRAM FOR ROAD SIDE FACILITIES

FIGURE 17: USE CASE DIAGRAM FOR TRAFFIC SIGNS
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, Iftode L., Dept. of CS, Univ. of Rutgers, “Location-aware services over vehicular ad-hoc networks using car to car communication”
INITIAL DESIGN REPORT

DIVISION OF LABOR

Vehicular Interface: Tugberk, Okan, Arif

Network: Anıl, Okan

GUI: Anıl, Arif

Simulators: Anıl, Tugberk

Map: Tugberk, Arif

GANTT CHART

<table>
<thead>
<tr>
<th>ID</th>
<th>Task Name</th>
<th>Start</th>
<th>Finish</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Revision of architectural design</td>
<td>21/12/2009</td>
<td>25/12/2009</td>
<td>1w</td>
</tr>
<tr>
<td>2</td>
<td>Prototype Coding</td>
<td>21/12/2009</td>
<td>04/01/2010</td>
<td>2.2w</td>
</tr>
<tr>
<td>3</td>
<td>GUI implementation</td>
<td>21/12/2009</td>
<td>04/01/2010</td>
<td>2.2w</td>
</tr>
<tr>
<td>4</td>
<td>Map implementation</td>
<td>21/12/2009</td>
<td>04/01/2010</td>
<td>2.2w</td>
</tr>
<tr>
<td>5</td>
<td>Network communication</td>
<td>21/12/2009</td>
<td>04/01/2010</td>
<td>2.2w</td>
</tr>
<tr>
<td>6</td>
<td>Prototype Testing</td>
<td>28/12/2009</td>
<td>04/01/2010</td>
<td>1.2w</td>
</tr>
<tr>
<td>7</td>
<td>Research on simulators</td>
<td>21/12/2009</td>
<td>01/01/2010</td>
<td>2w</td>
</tr>
<tr>
<td>8</td>
<td>Demo preparation</td>
<td>04/01/2010</td>
<td>08/01/2010</td>
<td>1w</td>
</tr>
<tr>
<td>9</td>
<td>Detailed Design Report Preparation</td>
<td>04/01/2010</td>
<td>08/01/2010</td>
<td>1w</td>
</tr>
<tr>
<td>10</td>
<td>Detailed Design Report Submission</td>
<td>08/01/2010</td>
<td>08/01/2010</td>
<td>0w</td>
</tr>
<tr>
<td>11</td>
<td>Prototype Demos</td>
<td>11/01/2010</td>
<td>22/01/2010</td>
<td>2w</td>
</tr>
</tbody>
</table>

FIGURE 18: GANTT CHART
1. Dikaikos D.M., Dept. of CS, Univ. of Cyprus, Florides A., Dept. of CS, Univ. of Cyprus, Nadeem T., Siemens Corporate Research, Iftode L., Dept. of CS, Univ. of Rutgers. Location aware services over vehicular ad hoc system using car to car communication.


