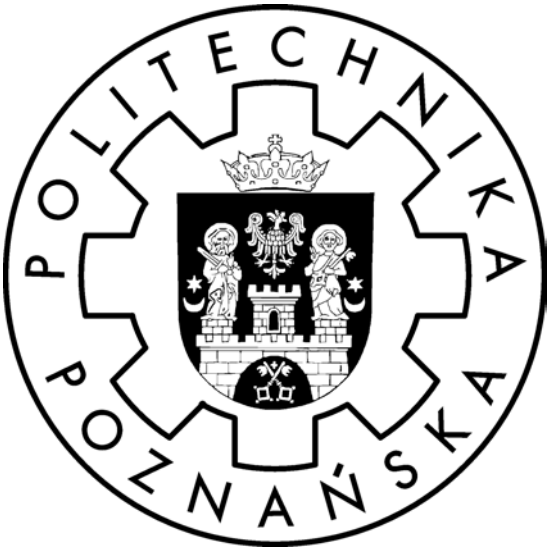
POLITECHNIKA POZNAŃSKA
INSTYTUT INFORMATYKI



„ZIELONY TUNEL”

SYSTEM WSPOMAGANIA RUCHU POJAZDÓW UPRZYWILEJOWANYCH

POZNAŃ 2003

Autorzy Bartosz Grabiec
Robert Guziołowski
Jędrzej Jajor
Marcin Zienkowicz

Promotor dr hab. inż. Jan Kniat

Koreferent dr hab. inż. Jerzy Nawrocki, prof. PP

This thesis is a Computer Society International Design Competition 2003 project. The goal of the competition is to advance excellence in computer science and computer engineering education by encouraging teams of students to design and implement computer-based solutions to real-world problems.

The Green Tunnel System – Mobile Emergency Services Support System project report is one of the parts of this work. The second part consists of appendices, which describe the implementation of the system in details.

The responsibility for creation of the system was divided as follows:

Bartosz Grabiec – overall system design, scheduling algorithms implementation, algorithms determining vehicle's fuzzy position.

Robert Guziołowski – overall system design, cryptography algorithms implementation, traffic lights controllers' software.

Jędrzej Jajor – overall system design, user interface implementation, communication protocols implementation.

Marcin Zienkowicz – overall system design, system database design, crossroads models, web portal maintenance.

Table of Contents

List of figures IX

CD contents XI

1 Abstract 3

2 System overview 5

2.1 Aims of the GREEN TUNNEL SYSTEM 5

2.2 Hardware and software requirements 7

3 System design & implementation 9

3.1 System design 9

3.2 GREEN TUNNEL SYSTEM base modules 11

3.2.1 Officer-on-Duty Module 11

3.2.2 Administration & Monitoring Modules 17

3.2.3 Green Tunnel Control Center 17

3.2.4 Vehicle Communication Module 25

3.2.5 Vehicle Module 27

3.2.6 Traffic Module 30

3.3 Tradeoffs 31

3.4 Developed and used tools 32

3.4.1 Traffic Simulator 32

3.4.2 Green Tunnel Module Monitor 32

3.4.3 GPS Emulator (GPSemu) 33

3.5 Data encryption and modules' authorization 34

4 Verification and testing 37

5 Summary 39

6 References 41

7 Appendices 43

7.1 Database description 43

7.1.1 Basic GREEN TUNNEL SYSTEM tables 43

7.1.2 GPS emulator tables 53

7.2 Database access classes 55

7.3 Security features 62

7.3.1 Security level and key sizes 62

7.3.2 Password file structure 62

7.3.3 RSA public and RSA private key structure 64

7.4 GPS Receiver Characteristics and Protocol 65

7.4.1 Receiver characteristics 65

7.4.2 Communication protocol 65

7.4.3 Hardware connection 66

7.4.4 NMEA settings for the RS232 67

7.4.5 Sample Sentence 67

7.5 Magnetoinductive compass characteristics 69

7.5.1 Compass characteristics 69

7.5.2 Compass communication protocol 69

7.5.3 Commands used in the Vehicle Module application 72

7.5.4 Hardware connection 73

7.5.5 Settings for RS 232 communication 74

7.5.6 Sample output word 75

7.6 The OBD-II – RS 232 interface characteristics 76
7.6.1 Circuit description 76
7.6.2 Interface schematics 76
7.6.3 Connection pinouts 78
7.6.4 OBD protocol description 78
7.6.5 Commands used in the Vehicle Module application 80
7.6.6 Settings for RS 232 communication 81
7.7 Serial port component 82
7.7.1 Component design & implementation 82
7.7.2 The CPortGPS subclass 85
7.7.3 The CPortTCM subclass 86
7.7.4 The CPortOBD subclass 87
7.7.5 The CPortRFM subclass 87
7.8 Messages exchanged between modules 88
7.8.1 Communication between the DAM and the ODM 88
7.8.2 Communication between the DAM and the GTM 96
7.8.3 Communication between the DAM and the TS 97
7.8.4 Communication between the DAM and the VCM 97
7.8.5 Communication between the CRM and the DAM 98
7.8.6 Communication between the VM and the VCM 99
7.9 Source code statistics 101
7.10 Glossary 103

List of figures

Figure 1 System architecture diagram. 6

Figure 2 Officer-on-Duty use case diagram 10

Figure 3 Vehicle Driver use case diagram 11

Figure 4 General User Interface design 12

Figure 5 The Request Dialog design 14

Figure 6 A screenshot of the Officer-on-Duty Module while defining a request 16

Figure 7 A part of Green Tunnel Database entity diagram 19

Figure 8 An exemplary graph representation 22

Figure 9 The Green Tunnel Module class diagram 23

Figure 10 The overview of the Vehicle Module 27

Figure 11 TCM 2-20 27

Figure 12 The Vehicle Module 29

Figure 13 The Vehicle Module's UI design and implementation 30

Figure 14 Green Tunnel Module Monitor main view 33

Figure 15 Establishing encrypted connection 35

Figure 16 The OBD-II - RS232 interface schematic 77

Figure 17 Serial port operation sequence diagram 84

Figure 18 Serial Port class diagram 85

CD contents

The enclosed CD-ROM contains:

- /source – contains source code of the GREEN TUNNEL SYSTEM modules,
- /install – contains the installation versions of the GREEN TUNNEL SYSTEM,
- /tools – contains other tools and programs (e.g. MySQL DBMS, ODBC Drivers, GPS Emu),
- /thesis – contains this thesis in Microsoft Word and Adobe PDF file formats

Poznan University of Technology
Wielkopolska, Poland

CSIDC 2003 - Final Report

Green Tunnel

Mobile Emergency Services Support System

Team members:

Bartosz Grabiec
brian@pro.onet.pl

Robert P. Guziolowski
goo_zik@poczta.onet.pl

Jedrzej Jajor
jedrzej@man.poznan.pl

Marcin Zienkowicz
marcas@man.poznan.pl

Team Mentor:
Jan Kniat
Jan.Kniat@cs.put.poznan.pl
+48616652124

1 Abstract

Success in a rescue operation is to a great extent conditional on how fast emergency services arrive at the place of an accident. Due to a constant increase in the number of cars in cities, many transportation problems arise. Traffic jams make fast movement of emergency services almost impossible. Facing the problems mentioned above we have worked out a comprehensive project of a system – The GREEN TUNNEL SYSTEM – that supports the functioning of emergency services in a city. The GREEN TUNNEL SYSTEM offers:

- registering accidents and monitoring them;
- localization of emergency vehicles in a city, automatic assigning them to accidents and collecting information about the state of each vehicle;
- computing the shortest path for the emergency vehicle taking the time of arrival at the place of an accident into consideration; the time of arrival is estimated on the basis of traffic intensity;
- setting up the Green Tunnels – „a wave" of green traffic lights at the crossroads located on the route of a vehicle.

The Green Tunnel considerably speeds up a drive of special vehicles through a city (in particular large fire vehicles, which cannot force their way through big traffic jams). Using an existing infrastructure of modern traffic lights controllers we gather data about traffic intensity on individual streets in the whole city. Thanks to it our system allows avoiding obstacles such as traffic congestions, and also sudden, unexpected road works or road blockades.

An automatic digital transmission of the appointed route to a vehicle avoids long time of "voice" data passing. Extra information about a call can be transmitted through a standard voice radio later. A visual presentation of the route on the screen located in the vehicle makes finding the destination easier without detailed knowledge of the city road network.

A driver has a possibility of informing the system of his vehicle state (available, unavailable, out of order, etc.) and obstacles which arise on his way. He can also request a Return Green Tunnel for his way back.

The typical scheme of an operation is as follows: the officer-on-duty is informed about an accident usually by phone. He defines the request and adds it to the GREEN TUNNEL SYSTEM. The system takes care of finding available emergency vehicles and informing their drivers about the route to the place of an accident. When the vehicle's route is known, the system affects the traffic lights controllers so that the vehicle need not wait for the green lights at subsequent crossroads. The controllers follow the local adaptation light controlling scheme, where the time of green and red lights depends on actual traffic intensity and priorities assigned to each of the directions. The effect of the Green Tunnel is achieved by assigning higher priority to the direction which is on the emergency vehicle's route.

The system is composed of one Administration & Monitoring Center, many Emergency Departments, one Green Tunnel Control Center and many Vehicle Modules installed in rescue vehicles.

2 System overview

2.1 Aims of the GREEN TUNNEL SYSTEM

The GREEN TUNNEL SYSTEM's task is to support emergency services in everyday work and critical situations. The system is composed of the following items: the Green Tunnel Control Center, the Administration & Monitoring Center, many Emergency Departments, the Vehicle Communication Station and many emergency vehicles. The Emergency Departments are located in such places as: Health Care Centers, City Fire Departments, Police Departments and so on. The Green Tunnel Control Center should be located in a convenient place, taken into consideration such factors as: the access to the traffic lights controllers' communication lines, access to the network, etc. The design assumes building one Vehicle Communication Station, so it should be placed possibly near the center of the area controlled by the system to achieve maximum effectiveness. The system's modularity enables us to change the design to support different model of communication, e.g. cellular network, in a simple way. The only difference would be in the protocol between the Vehicle Communication Station and the vehicles. The Administration & Monitoring Center can be located in one of the Emergency Departments, in The Green Tunnel Control Center or in other places, depending on the system's resources.

Special modules are provided for each of the mentioned places. A sample configuration of the system is presented in Figure 1.

The Officer-on-Duty Module is installed in every Emergency Department and is designed primarily for request management in the system. Three important types of its activities are: the receiving of requests to assign emergency vehicles to accidents and prepare the Green Tunnels for them, setting up a Permanent Green Tunnel on a certain route at a certain time, and sending the information about arising obstacles. The module's other possibilities include: the overview of the city (the traffic congestion, obstacles in the city, location of accidents, positions of emergency vehicles), vehicle management (adding, deleting and editing information about vehicles subject to the department), vehicle state control and user account management.

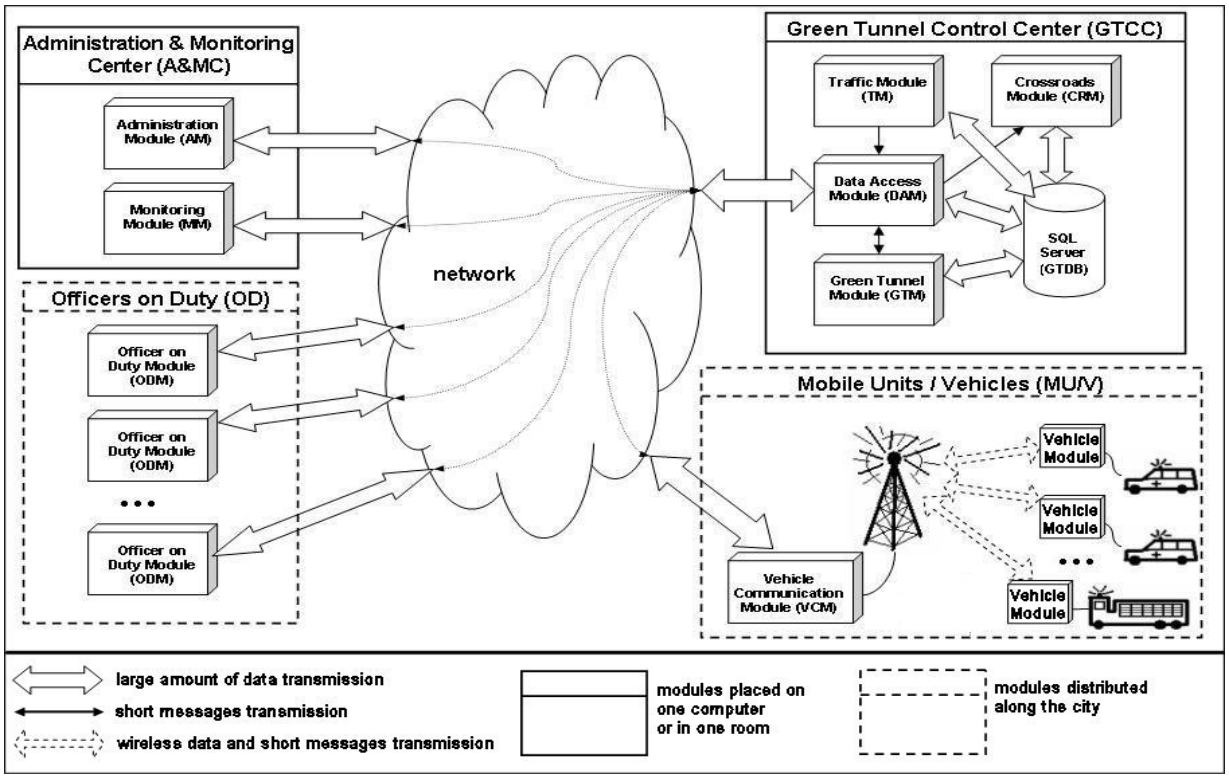


Figure 1 System architecture diagram.

The Green Tunnel Control Center is the core of the GREEN TUNNEL SYSTEM where all information is processed. The Green Tunnel Module decides which of the vehicles should be assigned to which task. Moreover, it computes the best route through the city for each of them, where the criterion of minimization is the time of arrival at the place of an accident. Then the information about the routes is passed on to the Vehicle Modules installed in all emergency vehicles through the Vehicle Communication Module. This information is also sent to individual traffic lights controllers through the Crossroads Module. The Crossroads Module ensures that the green light at each crossroad turns on as the vehicle approaches the crossroad, so that no time is wasted on waiting for the green light. The traffic light signaling is affected by changing the table of priorities for each traffic lights controller. Each of the vehicles assigned to an action is traced by the system – information about its position is gathered more frequently than from other vehicles to ensure that its drive through the city is fluent, no traffic congestion occurs on other roads and reaction to sudden changes in the traffic is as fast as possible. A current traffic

report is obtained periodically from the Traffic Control Center or directly from the sensors at the crossroads. The positions of individual vehicles are obtained on demand through the Vehicle Communication Module. The System Administrator located at the Administration & Monitoring Center supervises the work of all modules inside the Green Tunnel Control Center.

The driver of an emergency vehicle receives the information about the assigned task. The Vehicle Module displays the proper part of the city map with the suggested route plotted on it. After completing, he can send a request to set the Return Green Tunnel – a Green Tunnel from the place of an accident to the place the casualties should be taken to (e.g. a hospital). The driver can also report any obstacles in the city and the state of the vehicle.

As all the information processed in the system is of high importance, special means of security have to be used. It is ensured that unauthorized Emergency Department or vehicles can not establish connection to Green Tunnel Control Center and, therefore, can not send or receive crucial information about traffic intensity, accidents or vehicles' assignments to them. The possibility of wire-tapping is also limited as the transmitted data is encrypted with the use of Rijndael algorithm with automatically generated keys. The length of the keys can easily be changed in order to increase performance or fulfill exportation regulations of a given country.

2.2 Hardware and software requirements

The computers on which the GREEN TUNNEL SYSTEM's components are installed have to meet the following requirements. First and foremost all the computers working within the GREEN TUNNEL SYSTEM have to be linked via Internet or another similar network with a permanent public IP address. Furthermore, computers connected to the system need to have .NET Framework SDK. This in turn enables the system to be independent of a hardware platform and an operating system[1]. Computers designed to work at Emergency Departments can be any PC-compatible computers, capable of running Windows 98 or higher system, as the Officer's-on-Duty module

[1] At the moment of creating the system, the .NET Framework SDK was available only for Windows 98 and higher systems.

doesn't require any special resources. Necessary performance of computers in The Green Tunnel Control Center depends on the size of the system (i.e. number of Officer-on-Duty modules connected and estimated number of requests). To meet the scalability requirements each of the modules is designed to be able to work on a different computer to balance the load.

In addition a database management system with interface accessible through ODBC is required. The GREEN TUNNEL SYSTEM is a kind of distributed internet application and the communication model is a mixture of the message-passing model and the shared memory model.

3 System design & implementation

3.1 System design

In the process of creating the GREEN TUNNEL SYSTEM we employed a mix of waterfall software development model and eXtreme Programming. This strategy was chosen because it is adequate for unique research projects and speeds up the process of coding.

First of all we held interviews with potential users (e.g. police officers, fire fighters, ambulance drivers) and producers (e.g. producers of traffic light controllers) to learn the requirements the system should meet. Since we had no opportunity to have continuous contact with them we used a variant of Six Hats Technique (a kind of role playing) whenever difficult issues were to be solved. The information acquired during the interviews was essential in planning games we played. All that brought us to designing the system as a set of UML diagrams (with the use of Rational Rose).

The development process was divided into small iterations, which gave us an opportunity to control work progress and the process of achieving the design objectives as well. We made many spike solutions (small, informal experiments), as well as many modules' consolidations. It was very important since we planned to develop newly designed solutions and had to work with new technologies.

The testing process started simultaneously with development. All parts of the system were continuously tested. This all helped to detect errors early enough, when they were easy to locate and correct.

The implementation of the system is based upon a set of UseCase diagrams, created after interviewing its potential users. During the design phase we distinguished several actors:

a. Officer-on-duty – his role is to add and manage (that is: view, delete, edit) requests, which are to be processed by the system. He can also cancel some vehicles from an action. This is connected with canceling the Green Tunnel made for them, if it is no longer in use. The officer can also perform certain administrative tasks (such as defining new vehicles

that are subordinated to the place) and set up Permanent Green Tunnels. Furthermore, he is obliged to add information about arising obstacles to the system.

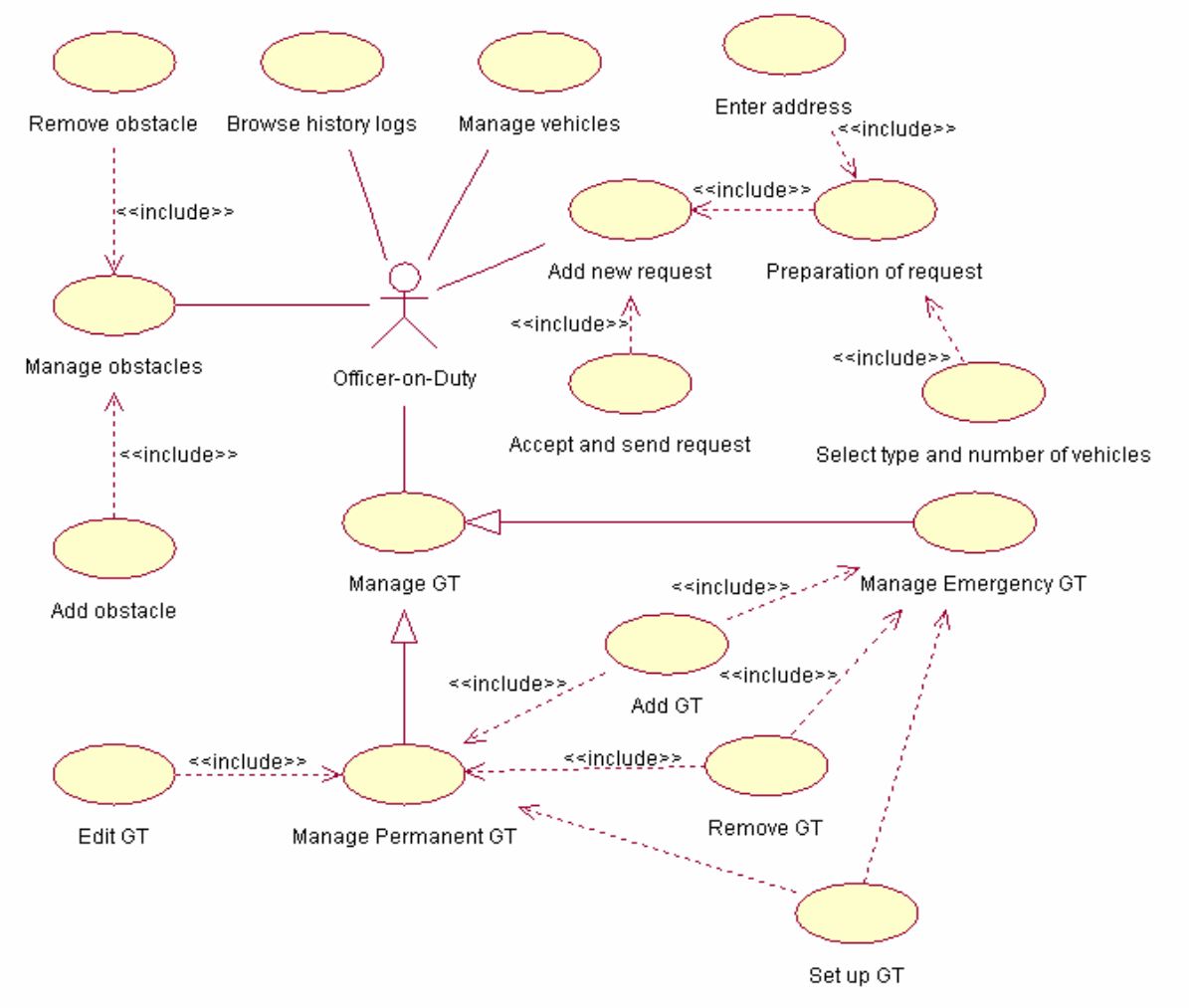


Figure 2 Officer-on-Duty use case diagram

b. System administrator is obliged to monitor the system continuously during its work. He can update the city map (add new streets, change their names etc.). It is also possible for him to check any time a vehicle position or traffic lights controller functioning. Another duty is to add the information about cooperating Emergency Departments. As a system maintainer he also takes care of its efficient functioning.

c. Vehicle Driver receives the suggested route plotted on the displayed city map. He can report the state of the vehicle, the fact of finding an obstacle on the route and send the request to set the Return Green Tunnel. The state of the vehicle is taken into account while assigning vehicles to requests.

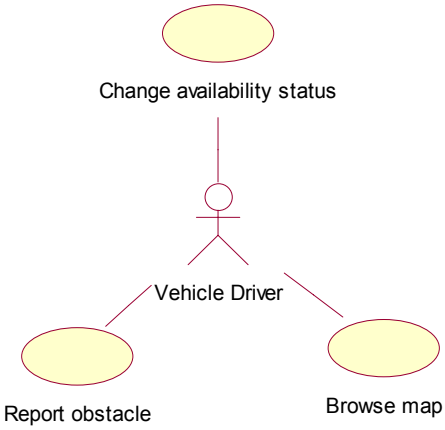


Figure 3 Vehicle Driver use case diagram

As the system development environment we have chosen Microsoft Visual Studio .NET. Mainly we have used C++ programming language.

3.2 GREEN TUNNEL SYSTEM *base modules*

3.2.1 Officer-on-Duty Module

The Officer-on-Duty Module is a system component installed on every Emergency Department computer. As said before, each of these computers is connected to the Internet or directly to the Green Tunnel Control Center using a dedicated link. The functionality offered by the module depends on the kind of institution it is installed in as well as the roles assigned to the logged person. This section covers the basic functions of the module: city overview, defining new requests, adding information about obstacles, defining Permanent Green Tunnels and administrative actions.

In order to decrease communication requirements, the software stores a local copy of the data from the central Green Tunnel Database. The data are downloaded from the database after establishing the first communication. Later the Data Access Module sends messages about data updates to all Officer-on-Duty Modules. This ensures data consistency, without superfluous link loading. The parts of the module exchange data using specially designed single-producer-multi-consumer thread-safe buffers. This gives the opportunity to add new parts of the module, without disturbing the work of others.

The separate question is the protection of the module against unauthorized access. For that purpose each user has his own login and password. The MD5 hash value of the user's password is computed. The supplied user's login and computed hash value of the password is sent to the Green Tunnel Control Center, where the process of verification is carried out (i.e. the sent hash value is compared to the value stored in the database for the user of a given login). Security issues impose that all messages transferred between the Officer-on-Duty Modules and the Green Tunnel Control Center through a potentially unsafe network are encrypted (see section 3.5). After the process of authorization is completed, the user can start his work with the module.

3.2.1.1 City view

The primary view the user is presented with is the city map, with the currently processed requests (the places of accidents and the Green Tunnels) and vehicle positions plotted on it. This provides the officer with a tool to evaluate the situation in the city.

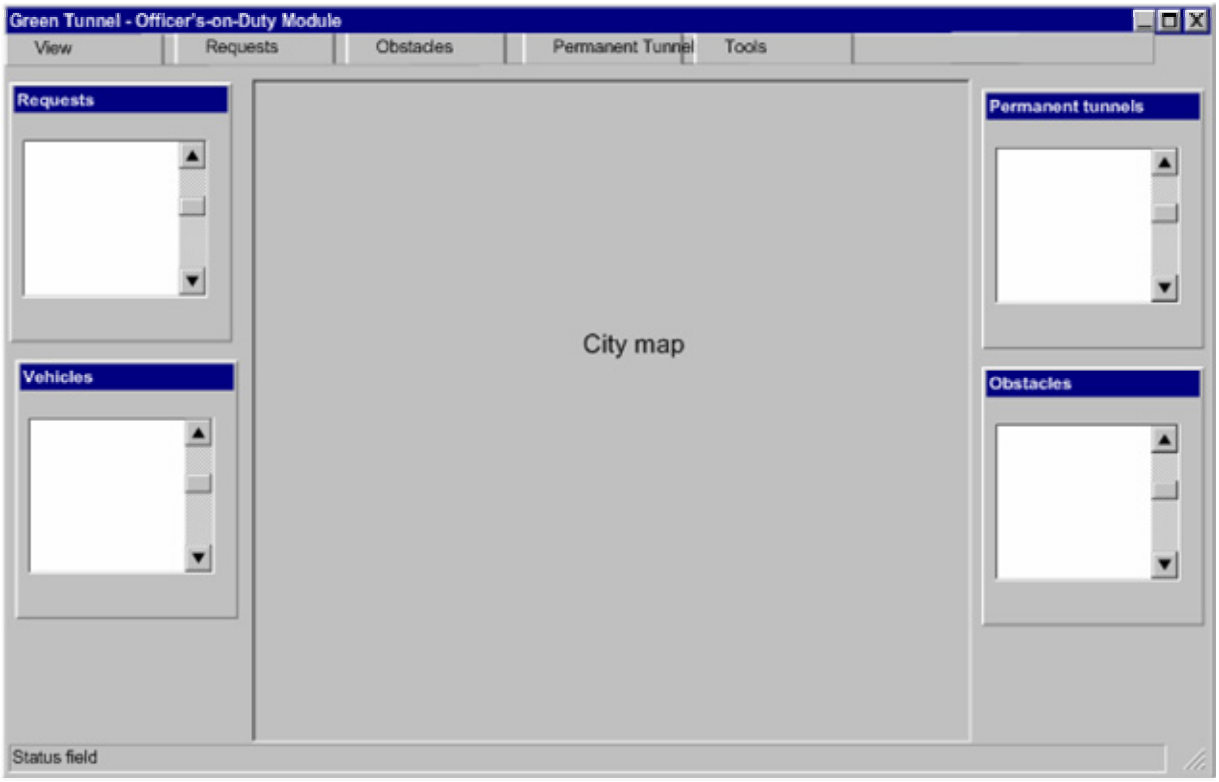


Figure 4 General User Interface design

The information shown on the map is easily to customize in order to provide all necessary details. This information may include:

- streets and crossroads in the city,
- additional characteristic objects,
- traffic state report (traffic congestion is presented using different colors),
- approximate positions of the vehicles,
- requests being processed,
- Green Tunnels set for particular requests,
- obstacles for the traffic in the city.

The scale of the map may be adjusted in a wide range to set the view on the chosen part of the city. Around the map, on the dialog bars, detailed information about all requests, vehicles, permanent Green Tunnels and obstacles is presented. Available functions can be run using menu commands (or associated buttons in toolbars, or keystrokes).

3.2.1.2 Defining a new request

The officer-on-duty can define new requests using the Request Wizard. The first page of the wizard is for entering the destination address. This can be done in two ways: by typing the address or by clicking the appropriate place on the map. The second page contains all available types of vehicles. The officer has to enter the number of vehicles of each type he wants to send to the place of an accident using spin buttons or by typing the numbers of vehicles of a given type. If vehicles of the requested type are unavailable at the moment (they were allocated to other requests or are unavailable for technical reasons), the system shows the officer a suitable message. He can select other vehicle types in place of the lacking ones. It is important that the officer does not assign individual vehicles, but only the type of the vehicles. The request is then processed in the Green Tunnel Control Center and the vehicles closest to the place of an accident are chosen. After adding a request by the officer, the list of requests visible on the main screen is refreshed, and a new symbol representing the place of the just-added

accident and a new Green Tunnel is shown on the map. Moreover, the list of vehicles contains the information about vehicles allocated to the request.

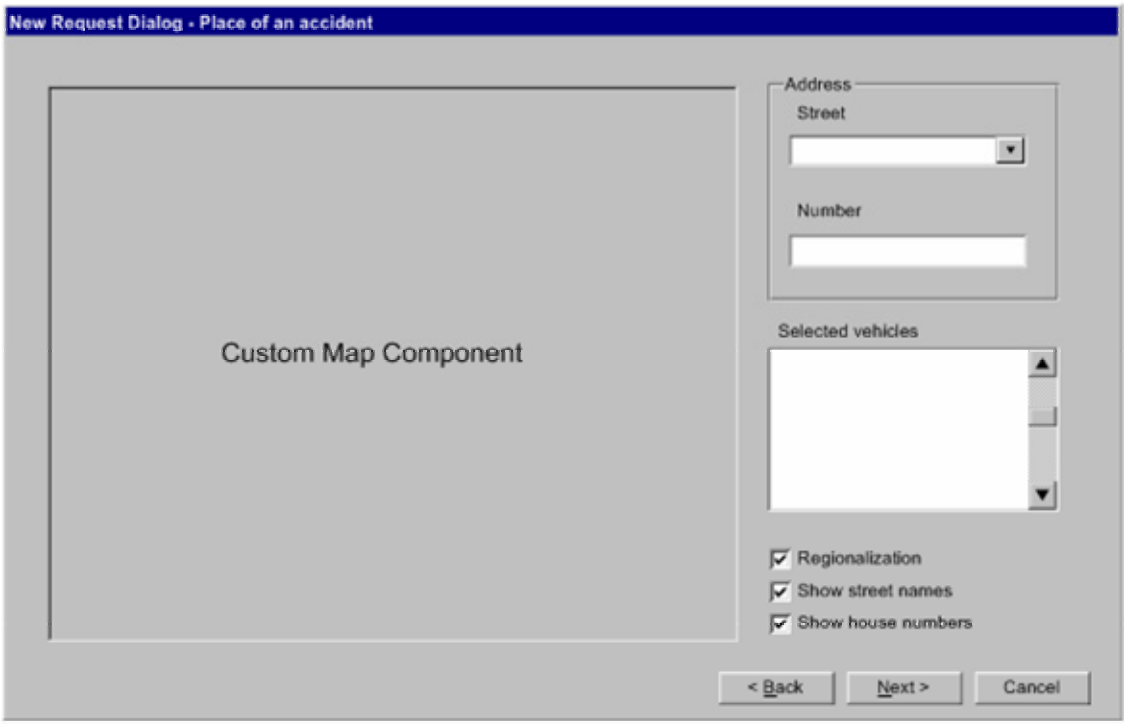


Figure 5 The Request Dialog design

3.2.1.3 Adding information about obstacles

Adding information about an obstacle resembles defining a request. Again, the officer is presented with the map and the list of street names, so that he can choose the points or areas of the obstacle occurrence. Additionally the start time (the default value is the moment of adding the obstacle) and optionally the end time (if it is known) can be supplied. Short description of the obstacle may also be attached.

The information gathered from the user is sent to the Green Tunnel Control Center, where all necessary actions are carried out. If the obstacle interferes with any of the Green Tunnels, the affected tunnels are recalculated.

The recalculation of the Green Tunnels is also forced when a message about removing the obstacle arrives at the system, because there is a chance that some of the tunnels could be set

on a shorter path. The process of allocating vehicles to requests itself is carried out in the Green Tunnel Module (see section 3.2.3.3) and sending data to certain traffic lights controllers is the domain of the Crossroads Module (section 3.2.3.4).

3.2.1.4 Adding Permanent Green Tunnels

Permanent Green Tunnels are designed for special-purpose transportation, where it is known in advance that a vehicle will follow a certain route at a certain time. The Officers-on-Duty Module allows the officer to add such Green Tunnels in several modes:

- known start and end points;

  In this case the officer enters the point at which the vehicle starts (using the map or a list of street names) and assigns a timestamp to it. Then he enters the point the vehicle is supposed to arrive at. After the officer sends this information to the Green Tunnel Control Center the system computes the best route for the vehicle taking into account the traffic congestion. The route can change due to new information about changes in the traffic or the arising of obstacles. In these cases the driver is informed about the change and can see the new route on his Vehicle Module.

- known start, end and some intermediate points;

  This is some modification of the above scheme. Here additional points in the city that the vehicle must pass are given. A timestamp can be associated with each point to represent the latest moment of reaching the point. The system computes the best route and displays it to the officer in order to confirm or reject. The route can change due to information about obstacles and the traffic, however all the supplied points on the route of the vehicle are taken into account while computing a new route.

- the whole route is given explicitly;

  The officer can manually choose the route for the vehicle, assigning a timestamp to every point on the route. The route can change only due to information about obstacles conflicting with the entered route.

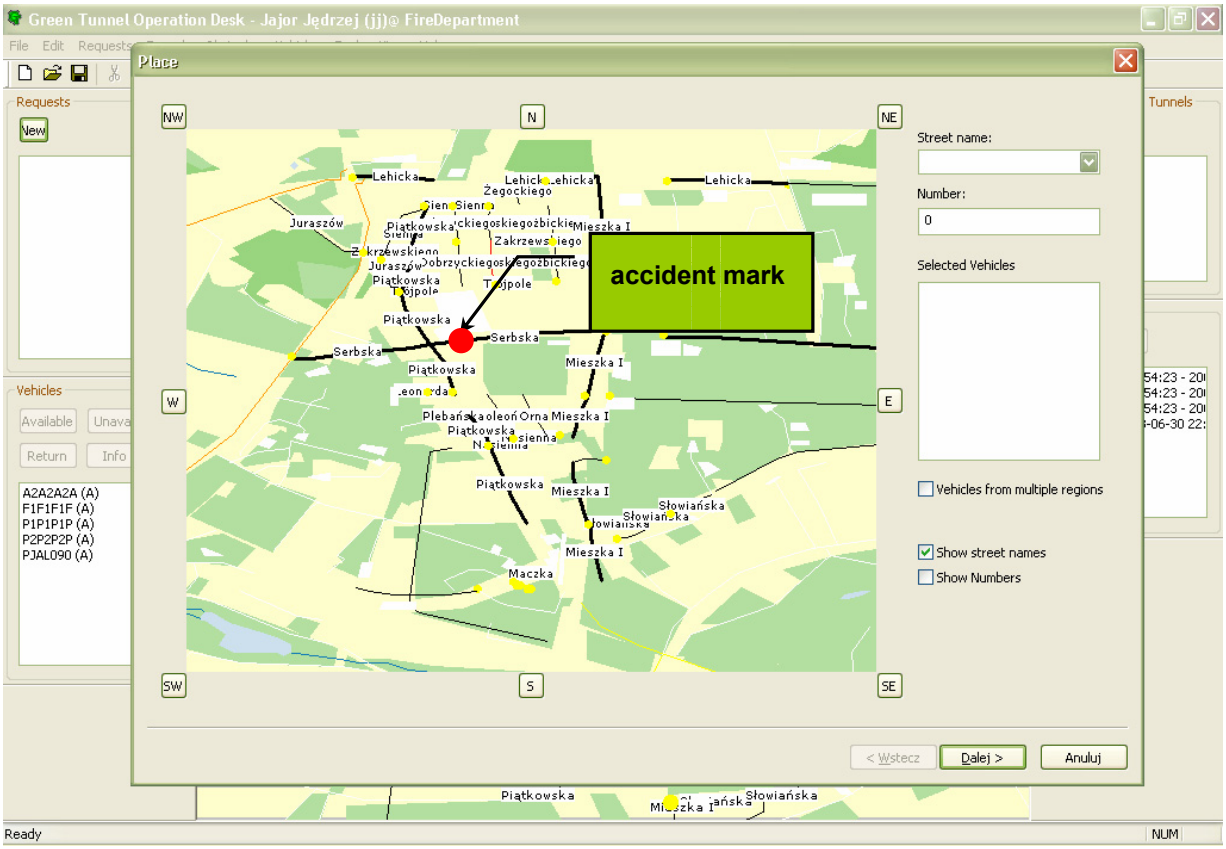


Figure 6 A screenshot of the Officer-on-Duty Module while defining a request

3.2.1.5 Administration

Administrative functions at the level of each Emergency Department comprise:

- adding, deleting and editing officers' accounts for persons working in a certain department;

  These functions are designed for the head of the department. He can grant any subset of his own privileges to any of his subordinates. The account of the head of the department is created by the GREEN TUNNEL SYSTEM Administrator at the Administration & Management Center.

- adding, deleting and editing information about vehicles subordinated to the station;

Officers who are granted appropriate privileges can perform all activities that concern management of the vehicles: defining new vehicle types, adding new vehicles etc.

3.2.2 Administration & Monitoring Modules

The Administration Module is provided to prepare the system to start working. The module provides a simple and convenient in use way to define the city map. The Emergency Departments that cooperate with the system can also be defined together with the detailed information about them and their locations. The next step is the addition of user accounts for the employees of the Monitoring Center and officers-on-duty, and granting the adequate privileges. Another important feature is the ability to generate keys for Emergency Departments' and vehicles' authorization process.

The Monitoring Module is a very important program which works all time during the GREEN TUNNEL SYSTEM activity. It monitors both the functioning of the system as a whole and the functioning of several modules. It also enables the administrator to diagnose traffic lights controllers and the Vehicles Modules. These tests are performed in the background and the administrator is alerted if any of them fails. The module also enables the administrator to check remotely the state of vehicles and traffic lights controllers. The system log files, which are accessible through the module, record all events in the system, especially any attempts of gaining unauthorized access or any failures.

During the work of the Monitoring Module a city plan illustrating current traffic situation is displayed. Additionally, positions of all vehicles operating in the GREEN TUNNEL SYSTEM and set up Green Tunnels are shown, too. It is also possible to show the list of accidents that take place in the city.

3.2.3 Green Tunnel Control Center

3.2.3.1 Green Tunnel Database

As a root of the Green Tunnel Database the MySQL database management system is used. The database is a crucial part of the GREEN TUNNEL SYSTEM, as its efficiency strongly influences

the efficiency of the whole system. The information that is stored in the database is the following: a city map with plans for every crossroad, requests, set up Green Tunnels, the last known positions of the vehicles, etc. The database stores also a part of the system log.

All Green Tunnel Control Center modules use the database through direct connections via ODBC, but no encryption is needed as these components are installed on one computer or in a safe local network without any connection with outer network, such as the Internet.

Because some modules (e.g. Officers-on-Duty Module) communicate with the Green Tunnel Control Center via a network (e.g. the Internet) we decided to use encrypted messages (for encryption method see 3.5). These messages are accepted and processed by the Data Access Module (see 3.2.3.2).

A part of the entity diagram of the Green Tunnel Database (modeled in Rational Rose as a class diagram) is presented in Figure 7.

3.2.3.2 Data Access Module

The Data Access Module is a component which makes all the communication between the Green Tunnel Control Center and other parts of the system possible. This is the place where outgoing information is encrypted and sent to other modules over the network. Here the data from other modules is also received. Every change in the global data is written to the database, and it is the Data Access Module's job to inform other modules about an update. Another important task of this module is an initiation of Green Tunnels recalculation whenever a change in the traffic is detected, a colliding obstacle appeared or was removed, or a vehicle became available or unavailable.

The Data Access Module has to fulfill some basic requirements, which are:

- module availability and immediate reaction for requests and incoming connections (every request or connection must be forthright received and served);
- high reliability and stability, as the Data Access Module is a central part of the system – any DAM failure implies a failure of the whole GREEN TUNNEL SYSTEM;

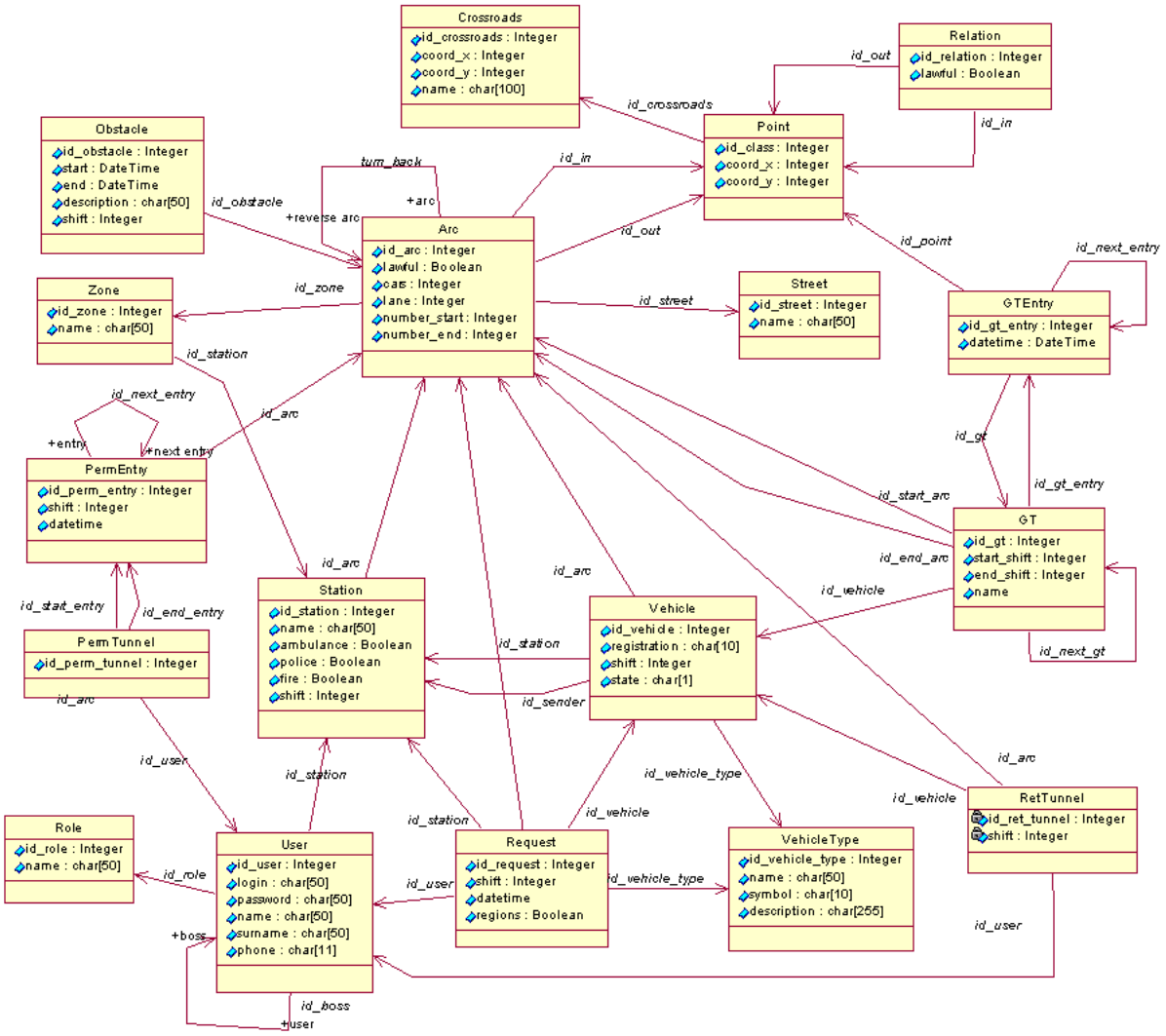


Figure 7 A part of Green Tunnel Database entity diagram.

- high performance;
- recording in system logs all operations, which would be helpful in reconstructing the course of events in case of a system failure, catastrophe, or other states of emergency).

In particular, the Data Access Module should provide a possibility to connect and communicate with:

- many Officer-on-Duty Modules (see section 3.2.1);

- one Vehicle Communication Module (see section 3.2.4);
- one Green Tunnel Module (see section 3.2.3.3);
- one Traffic Module (see section 3.2.6) or Traffic Simulator (see section 3.4.1).

To ensure the desired level of security, all transmitted data are enciphered by means of Rijndael algorithm, and the process of data exchanging starts after successful procedure of modules authorization.

Basic requirements of availability, immediate reaction, and high performance have been accomplished with the use of a multithreaded architecture. It is possible for other modules to connect simultaneously due to a listening thread that is always active. After establishing a connection a separate thread is started to serve it, i.e. receive requests from outer module, obtain answers form the Green Tunnel Database, and send them back. All requests are processed in the FIFO order. High performance of this module is related with the possibility of paralleling the task processed in separate threads. It can be also increased by the hardware support for parallel processing. Naturally, special measures of precaution were used to ensure a thread-safe operation, e.g. mutual exclusion using semaphores or monitors. High reliability has been obtained by many hours of testing, simulations and performance tests. A system log is run with the use of computer's local time. A registry is made immediate after successfully realized task. A fragment of this log is shown below.

```
2003-07-09 10:33:52 start
2003-07-09 10:33:52 listen start [IP:127.0.0.1:5999]
2003-07-09 10:33:52 connect [IP:127.0.0.1:6003]
2003-07-09 10:33:53 receive message:LoginVCM [name:vcm] [password:***]
2003-07-09 10:33:53 send message: LoginVCMAck
```

Communication with the Green Tunnel Database is made with the help of ODBC drivers. Message passing between modules is realized with the use of encrypted sockets, which are described in section 3.5.

3.2.3.3 Green Tunnel Module

The algorithms in the Green Tunnel Module process continuously flooding data about the states of the vehicles and traffic intensity to calculate a scheme of routes for specific Green Tunnels. In systems such as the GREEN TUNNEL SYSTEM, it is known that fulfillment of the performance criteria is not so easy, mainly because of the fact that cohesion of information depends on many factors e.g.:

- computer network – its reliability, connection time, speed etc.;
- people – drivers of the monitored vehicles and operators of the system, who receive and manage events;
- the system indeed – its failures and ability to react to both expected and unexpected changes.

As it was mentioned above there are three kinds of Green Tunnel implemented at present. Each of them has its own data, which is gathered from the main database and after adequate processing Green Tunnels are calculated. It is important to mention that the permanent Green Tunnels are represented in the database by a number of smaller Green Tunnels accordingly to midpoints of the tunnel.

Each of the vehicles in the system sends periodically information about its position. Just after receiving the information, the Green Tunnel Control Center knows the exact position of the vehicle. Unfortunately, up to the next transmission the position is unknown and uncertain. Then, knowing the last coordinates of the vehicle, its speed and direction of the movement, the Green Tunnel Module calculates its approximate, fuzzy position in the city [1]. Shortly speaking, the process consists in determining the area where the vehicle probably could be – the Green Tunnel Module is adapted to computing the possible range and then takes into consideration the city plan. The obtained set of streets and intersections for every vehicle is used in further stages of setting the Green Tunnels.

In the first place, during the design phase of the module, we chose the data structure for storing the city plan. The structure of the edge-weighted graph, which was used here, consists of two

hash tables: a table of vertices and a table of arcs. Each of the vertices represents intersection or its part, whereas the arcs denote set of parallel running lanes in a given direction. Since both of the tables use unique identifiers of the crossroads and streets, they allow us to find a representation of any place of the city in a relatively short time. Additionally, we store Cartesian coordinates of crossroads in the graph. Such data are required to determine the positions of all vehicles on the map and distances between objects i.e. intersections, cars etc. Due to minimization of searching and associating concrete vertices of the graph with coordinates from a navigational device, the list of vertices got an extra index, which can sort them on the coordinates (see Figure 8).

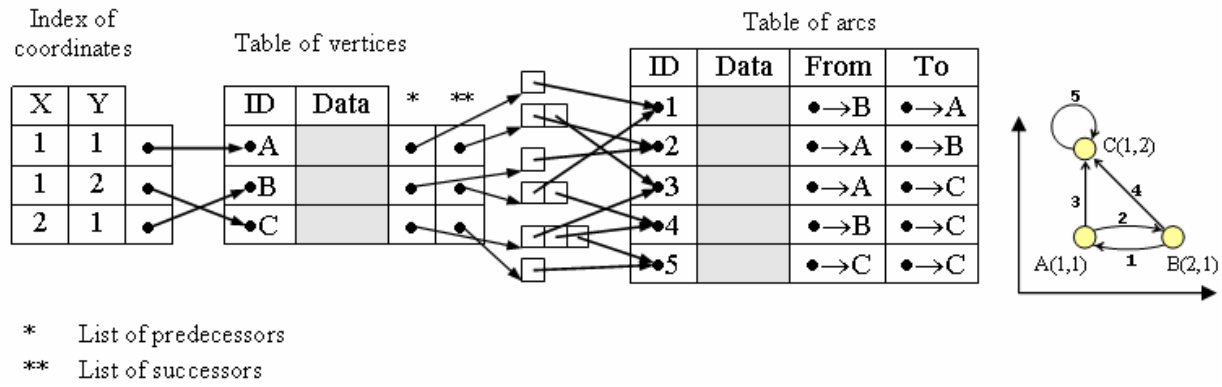


Figure 8 An exemplary graph representation

After getting information about location of an emergency vehicle, its state (i.e. whether it is in working order or it could meet obstacles on its way or maybe it is currently in a rescue action) and the place of accident, the Green Tunnel Module can assign it to the event and calculate the minimal time route to the target.

To find the shortest path for emergency vehicles we use Dijkstra's algorithm [2][3]. What is worth mentioning is the complexity of the algorithm. For the given data structure the complexity is $O((V+E)\log V)$ (where V – number of vertices, E – number of arcs).

We have to note that there exists a possibility of intersection of the Green Tunnels. However, it does not interfere with anything, because a crossroad where two or more tunnels intersect can still work without collisions.

The Green Tunnel Module accepts triggering of tunnel recalculations from the Data Access Module. There is a protection against too frequent recalculation, which could overload the Green Tunnel Module.

Data produced by the Green Tunnel Module are used mostly by the Crossroads Module (see section 3.2.3.4), which manages traffic lights controllers in the city. It is also necessary to stress that Green Tunnel module is universal and prepared for future development.

The complete class diagram of the module is presented in Figure 9.

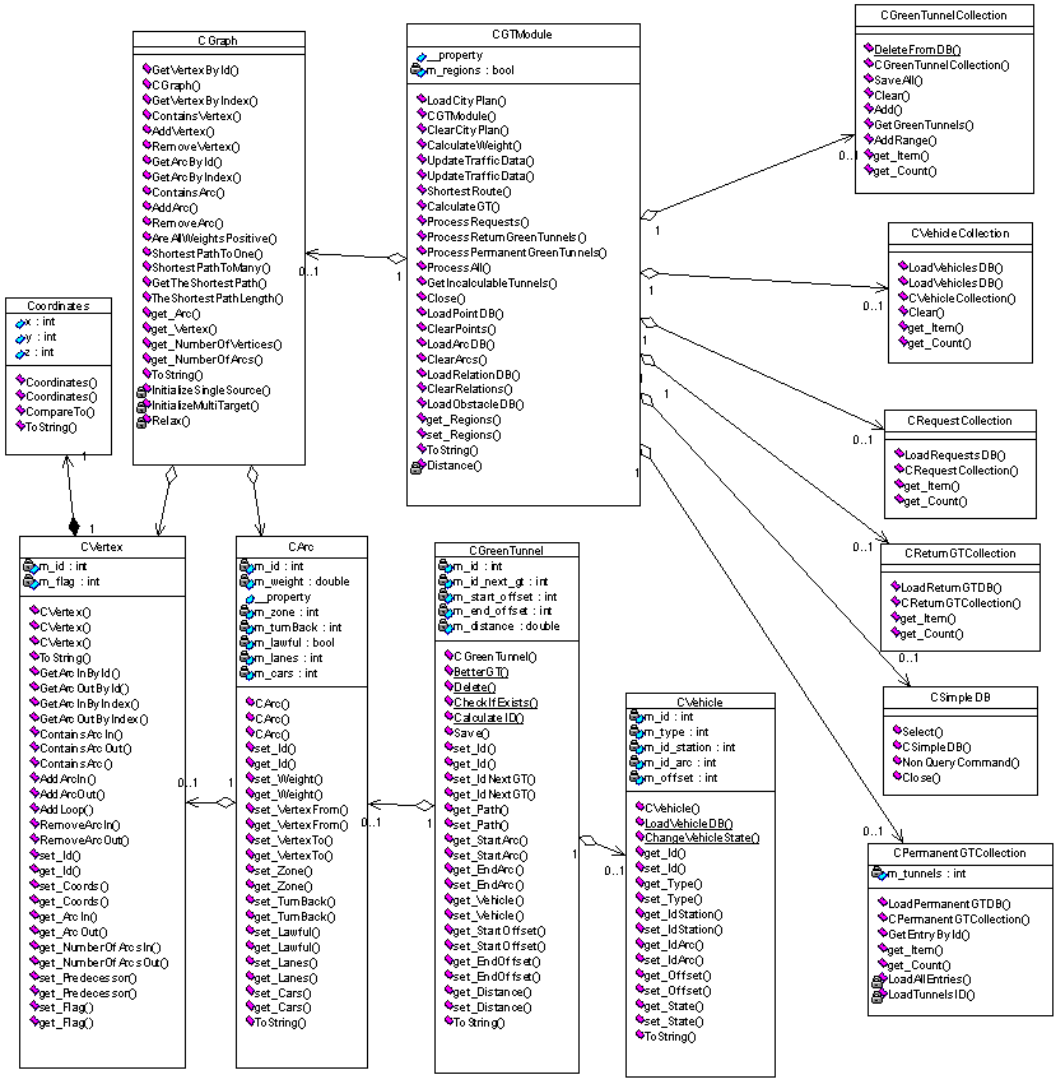


Figure 9 The Green Tunnel Module class diagram

3.2.3.4 Crossroads Module

The Green Tunnel Module determines the Green Tunnels in the city. Still, they have to be set on the real crossroads. This task is done by another module called the Crossroads Module.

Before we describe a process of a Green Tunnel creation, we have to consider some issues. First of all, it is unacceptable to switch on a constant green light on the affected direction. This would definitely lead to blocking all remaining directions or, in the optimistic case, drivers would think the traffic lights are out of order and drive using the rules of the road. Another issue is that leaving normal traffic lights controller's program running and turning green light on the desired direction on is also a bad idea – colliding directions would have green light on and soon whole intersection would be blocked with crashed cars. Determining the program for the traffic lights controller *ad hoc* has a disadvantage in high computation power consumption, so it is not a good answer, too.

The solution emerges in the architecture of modern traffic lights controllers. As they adapt locally to the junctions' traffic congestion, the embedded programs secure traffic unloading. They also provide us with the possibility of changing each direction's priority remotely, so the problem is solved – all that needs to be done is simply to increase the priority of the desired direction and the traffic light controller will do the job.

There are two stages in functioning of the Crossroads Module: the first one is connected with setting up a completely new tunnel that had just emerged in the Green Tunnel Database while the second one concerns maintaining the existing tunnels. Both stages will be discussed below.

Setting up a green tunnel begins with the gathering all the necessary information. The Data Access Module informs that there is a new tunnel waiting for being set on crossroads. Then the Crossroads Module connects with the Green Tunnel Database directly (if both, the module and the database, are in the Green Tunnel Control Center) or indirectly (with the use of the Data Access Module). It downloads the data concerning the definitions of new green tunnels and current traffic intensity. We assume that the definitions of junctions, their directions and the traffic light controllers' positions have been downloaded when the GREEN TUNNEL SYSTEM was starting up. The first green light is set on the first affected junction, as the reaction of the system

should be as fast as possible. (It is necessary to notice that we discuss here the setting up of a Green Tunnel, not a Permanent or Return one. We will discuss them later.) After that, the Crossroads Module ends the first stage and begins maintaining the existing tunnel.

The second stage requires the cooperation with the Data Access Module, the Vehicle Communication Module and the affected vehicle's Vehicle Module. As the Green Tunnel Modules determines all timestamps for junctions in the Green Tunnel, these timestamps may soon become useless. It is mainly caused by the change in the traffic intensity and the change in vehicles' positions. The change of the traffic volume causes the Green Tunnel Module to recalculate all existing Green Tunnels. But the position of the vehicle affects only the specified Green Tunnel. Thus, the Crossroads Module traces all the vehicles that are following their Green Tunnels asking them for their position more frequently then if they are not assigned to any action. This ensures switching the green lights on the junctions on time (taking into consideration the period of time necessary to unload the direction's traffic), as well as restoring usual traffic lights controller's program as the vehicle passes the intersection.

Setting up Return Green Tunnels does not differ from the solution described above. But we cannot say that about Permanent Green Tunnels. Even though, changes in calculations are not so significant, as a Permanent Green Tunnel can be seen as a mixture of a "usual" Green Tunnel and a Green Tunnel with already calculated timestamps (they will not change).

Unfortunately, none of the local traffic lights controllers' producer (i.e. MSR Traffic or NH-Polska) wanted to cooperate with our team. Thus it was impossible to obtain a transmission protocol specification used to remotely program of the controllers. Therefore the Crossroad Module does not implement any outgoing communication, but the appropriate priority changes on the junctions are printed in the module's console window.

3.2.4 Vehicle Communication Module

The use of the mobile components in the system imposes the need to build a station which is an access point for the wireless network. The design assumed that one base station should be built – a Vehicle Communication Station – in a central point of the area controlled by the GREEN TUNNEL SYSTEM. The advantage of this solution is cutting the costs of building

the dedicated networks of stations of another type (e.g. cellular). But there is also a disadvantage – the need to run the transmitter of relatively high power (dependent on the area). Particular communication system should be chosen dependent on the place of implementation of the GREEN TUNNEL SYSTEM. In the prototype small RFM 433 MHz modules are used [5].

The Vehicle Communication Module has to fulfill some basic requirements, which are:

- availability and immediate reaction for requests (every request or connection must be forthright received and served);
- high reliability and stability (a failure of this module implies the loss of connectivity with all the vehicles);
- recording in logs all operations and events (a kind of black box for vehicles).

In particular, the Vehicle Communication Module should provide a possibility to connect and communicate with:

- many Vehicle Modules (see section 3.2.5);
- one Data Access Module (see section 3.2.3.2).

Basic requirements of availability and immediate reaction have been accomplished with the use of a multithreaded architecture. After establishing a connection with the Data Access Module, a separate thread is started to serve it, i.e. receive data from Data Access Module and transmit (broadcast) them to Vehicle Modules. Each communication with a vehicle is also served in a separate thread. With the use of hardware support for parallel processing the performance of the Vehicle Communication Module can be easily increased. The security of transmitted data is provided by the process of each vehicle's authorization and encryption, discussed in section 3.5. A system log is run with the use of computer's local time. A registry is made immediate after a successfully completed task. A fragment of this log is shown below.

```
2003-07-09 10:33:52 connect [IP: 127.0.0.1:5999]
2003-07-09 10:33:53 send message: LoginVCM [name: vcm] [password: ***]
2003-07-09 10:33:53 receive message: LoginVCMAck
2003-07-09 10:33:53 listen DAM start
2003-07-09 10:33:53 listen VM start [IP: 127.0.0.1:6001]
```

3.2.5 Vehicle Module

In this and the next section we describe the hardware part of the project – the equipment of all rescue vehicles – with special emphasis on the vehicle position measurement system, computer and user interface.

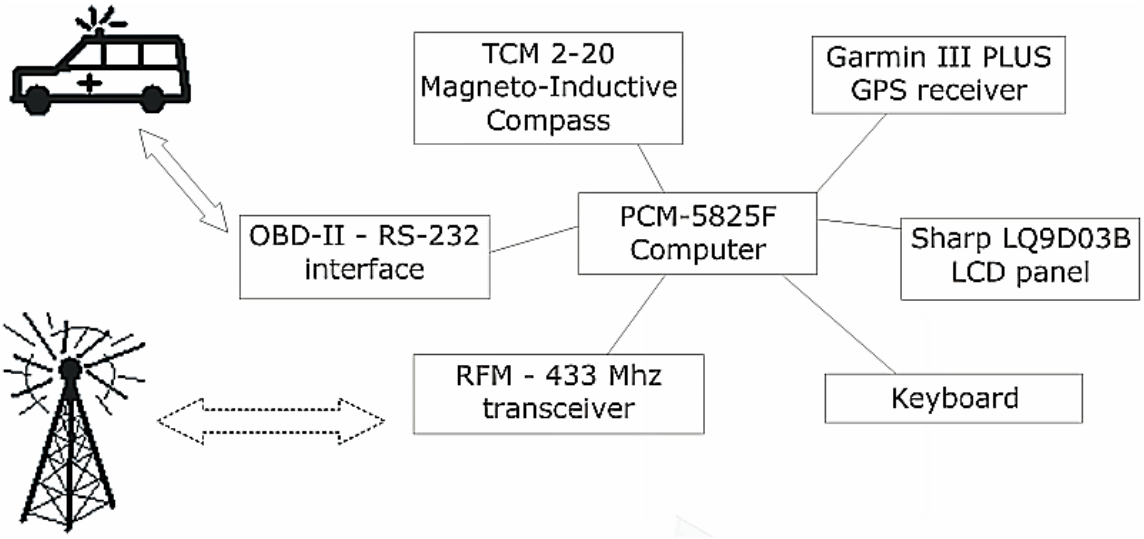


Figure 10 The overview of the Vehicle Module

3.2.5.1 Vehicle position measurement system

In order to be able to localize the vehicles we decided to use the Global Positioning System (GPS), because it is very popular, can be used without any charge and the receivers can be purchased at a reasonable price. As it is known, the GPS satellite signal may be inaccessible among high buildings, so the device is designed in such a way that the position of the vehicle could be obtained even if the GPS signal is inaccessible or corrupted.

The GPS receiver measures the position when GPS signal is available. Additionally works another system based on the geographical direction of a vehicle movement (measured by the digital compass shown in Figure 11) and vehicle speed data obtained from the vehicle's on-board computer through the OBD-II diagnostic link. During normal operation the GPS and compass data are averaged to minimize GPS signal interferences. When the GPS

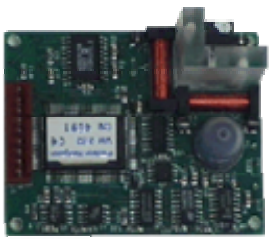


Figure 11 TCM 2-20

signal is lost, current position is computed as the last known GPS position with the displacement calculated from the movement direction and vehicle speed. The resulting geographical position is converted to the form (*arc_id*, *offset*), where *arc_id* is the identifier of the street and the *offset* is the distance between the beginning of the street and the actual vehicle position (see 3.2.3.1 for details). Such format is used in other components of the system. In the prototype we used the Magneto-Inductive Compass TCM 2-20 and the Garmin III PLUS GPS receiver [6][7].

3.2.5.2 The Vehicle Module computer

As the main computer in the vehicle module we have chosen the PCM-5825F biscuit PC with 300 MHz CPU and 128 MB RAM. This module, equipped with three RS232 ports, one RS232/422/485 port and a parallel port enables the connection of other components [8]. One of the ports is used by the GPS receiver. Another one is connected to the vehicle's OBD-II connector through the appropriate interface. Moreover, the computer can directly drive an 18-bit TFT LCD panel [9]. This choice was also partly imposed by the equipment we have in our laboratory, granted by the IEEE Computer Society as the Project Kit for the first edition of CSIDC.

The display shows the map centered accordingly to the current position of the vehicle. On the map the information about the computed route is plotted. Next to the map detailed information about the request is displayed (the address and other supplied data). The Vehicle Module enables the driver to send information concerning the vehicle state (available/unavailable/broken etc.) or arising obstacles to the Green Tunnel Control Center. This information is passed to the officer in the appropriate Emergency Department, who will enter certain messages to the system. The driver can also send a request to set the Return Green Tunnel for the vehicle. The Return Green Tunnel is set from the place of an accident to the place, where the emergency vehicle is heading (usually it is a hospital, where the casualties are taken). The request is passed to the officer who added the information about the accident to the system and he has to decide about the end point of the Return Tunnel.

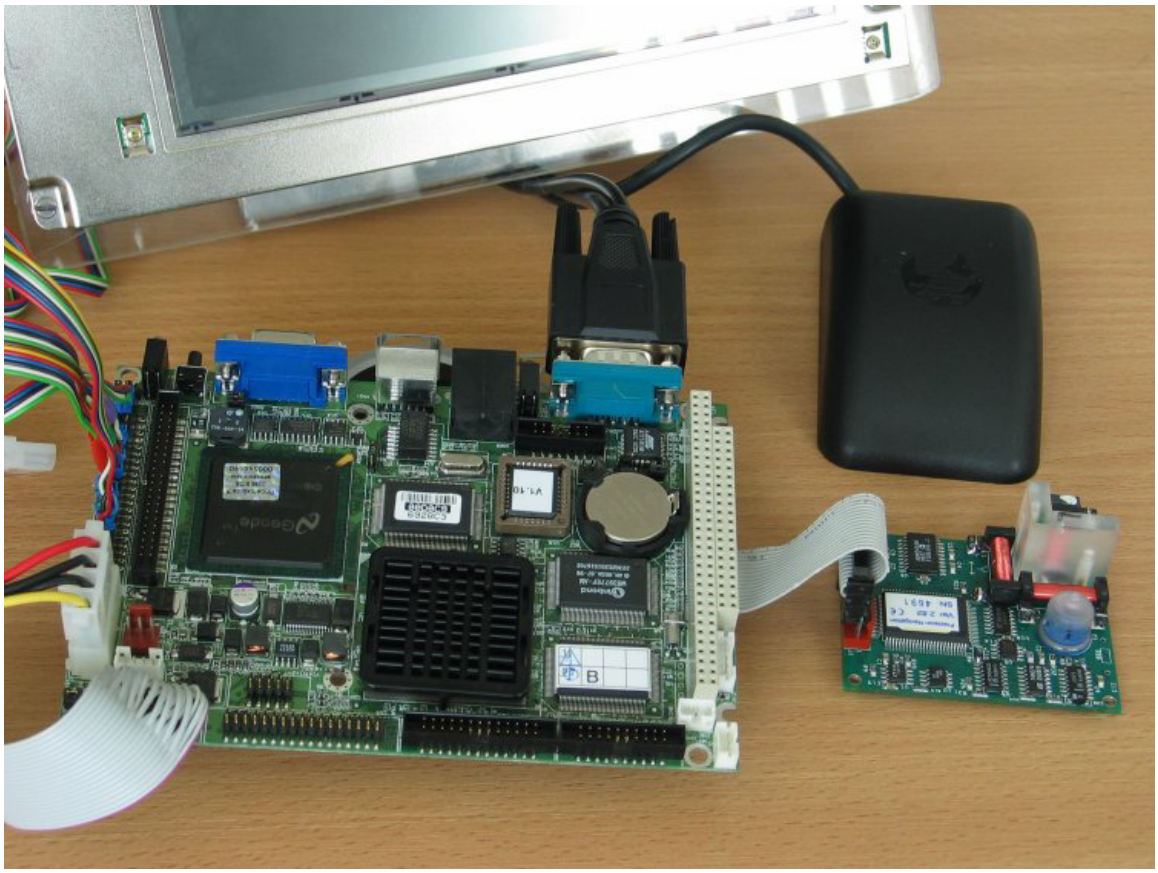


Figure 12 The Vehicle Module

It would be ideal to use a touch panel in order to create a user-friendly module; however this solution is rather expensive. Instead, a small keyboard, or device that simulates the behavior of a touch-screen can be used. The operation principle of such kind of equipment bases on interrupting the beam of infrared light. Whatever solution is chosen, it must be comfortable to read the data without excessive absorption of the driver's attention.

3.2.5.3 Vehicle computer software

The Vehicle Module computer is equipped with 256 MB SanDisk CompactFlash disk. This allows installing the Windows 98 operating system and the .NET Framework SDK. The program uses a 640x480 graphics mode, which is sufficient for displaying the city map and several button images. The data needed to draw a map are stored on the disk in a file, as the road network does not often change.

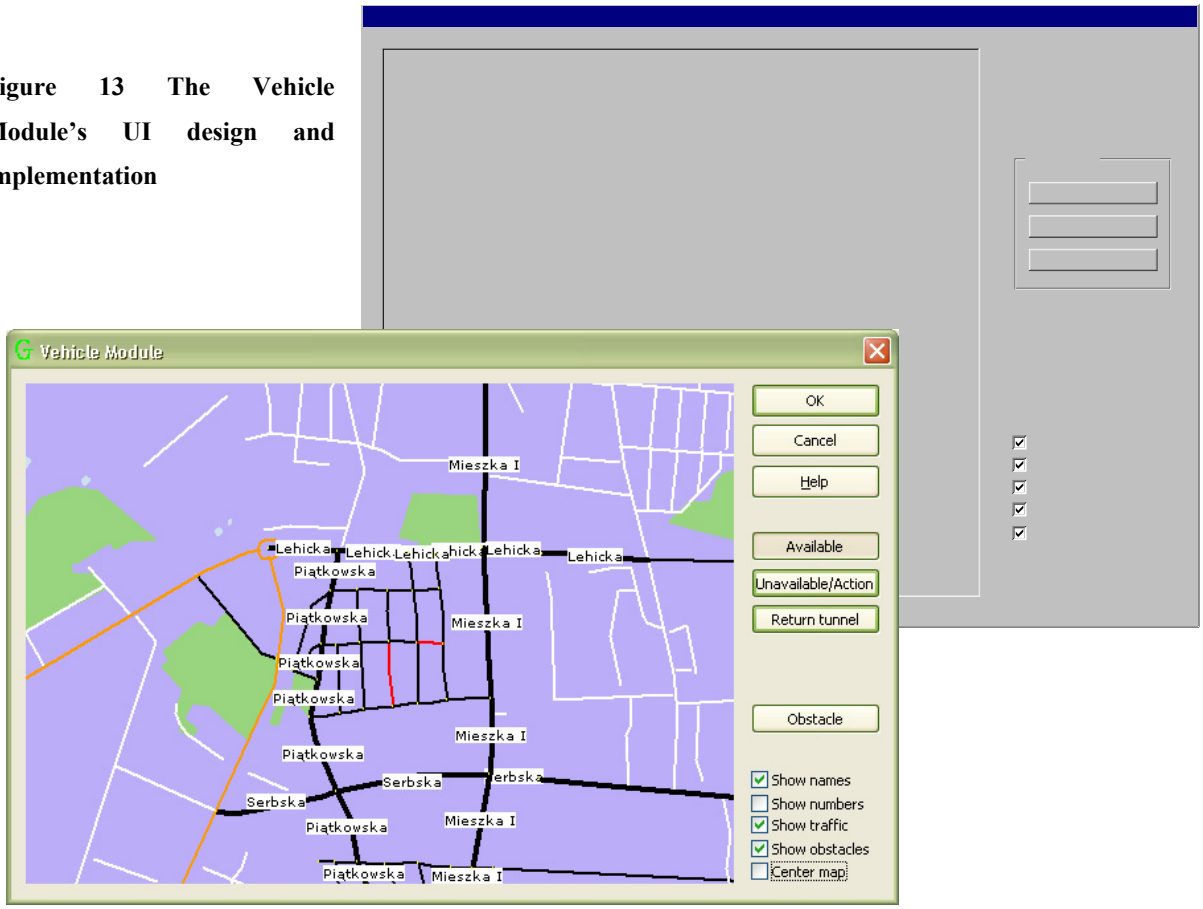


Figure 13 The Vehicle Module's UI design and implementation

3.2.6 Traffic Module

The GREEN TUNNEL SYSTEM gathers data concerning traffic intensity from many different sources (directly from crossroads or from the Traffic Control Center). This information can be sent in various formats, hence there is a need of a simple module which will convert it to the format used in the GREEN TUNNEL SYSTEM. The Traffic Module has a possibility to change the format of the data that comes from the Poznań Traffic Control Center. The traffic report is usually gathered every 10 minutes, but the sampling time is the Administrator's choice.

3.3 Tradeoffs

During the process of implementation several important decisions had to be made. One of them concerned the traffic lights steering. We had three options:

- manual setting the green and red lights for each direction;

  This solution requires real-time control at every crossroad, so a lot of computational power is needed and the existing traffic lights controllers are almost not used. Care must be taken not to open colliding directions.

- changing the length of the maximum and minimum green light times for each direction;

  This is a much safer solution, as the existing traffic lights controllers takes care of the actual green and red light steering, according to maximum and minimum green light times, which can be set only in the allowed range. Less computational power is needed, however still the new max and min times must be calculated and sent to every controller.

- assigning a priority to each direction;

  This is also a safe solution, as the controllers determine the maximum and minimum green light times based on the assigned priority. Very little computational power is needed, as only the new priority is transmitted to the controller. When the Green Tunnel is no longer needed, the priority is re-set to the default value for the crossroad.

The Crossroads Module implements the third of the schemes presented above, as it requires little resources and makes full use of modern traffic lights controllers (see section 3.2.3.4).

The algorithm chosen for the shortest path computing has great influence on system performance. Dijkstra's algorithm assumes that all arcs have positive costs as it is in the graph representing a city plan. Bellman-Ford algorithm is more general (can work with negative weights) but is less efficient. The other thing is that a number of Green Tunnels at a specific time is far less than a number of the shortest paths between all vertices. So Dijkstra's algorithm can be used *m* times (where *m* denotes a number of Green Tunnels) instead of using special algorithms e.g. Floyd-Warshall's algorithm or Johnson's one for all-pairs shortest path problem.

3.4 Developed and used tools

For the need of system implementation and partially for testing, we made use of several additional programs and software tools, which were developed by us or our friends.

3.4.1 Traffic Simulator

The Traffic Simulator, developed by our team, is a program which simulates the traffic intensity when no data from crossroads or the Traffic Control Center is accessible. It bases on a city plan, the last known traffic state in the city and the existing obstacles obtained from the Green Tunnel Database. Having all these information, the Traffic Simulator generates traffic intensity in the whole city after the specified time quantum, using the *group model*, which describes vehicles' behavior.

The Traffic Simulator is fully configurable. It is possible to set values of the factors typical for the specified country, such as average speed, maximum legal speed, etc. The length of time quantum of the traffic intensity simulation can also be changed.

Having studied much information from traffic sensors placed at crossroads, we decided to use 5 minutes time quantum. This value ensures the desirable traffic intensity accuracy and it does not overload communication infrastructure.

We had to develop the Traffic Simulator because:

1) we had only limited access to data from the Poznan Traffic Control Center – thanks to the Traffic Simulator we had our data available all the time,
2) failures of inductive loops and other sensors at crossroads occurred – it is also possible to simulate the traffic in the places of failure with the Traffic Simulator.

3.4.2 Green Tunnel Module Monitor

Green Tunnel Module Monitor is the additional program to Green Tunnel Module. This program has a simple graphical user interface and enables its user to see the current basic parameters of the main database. It was chiefly written for the testing phase but can be use during normal functioning of the Green Tunnel system as well.

Among the data, which could be gather from the program are: directions of specific roads and their intensity, ID of all arcs and crossroads, routes of the green tunnels, positions of accidents taking current database state into consideration.

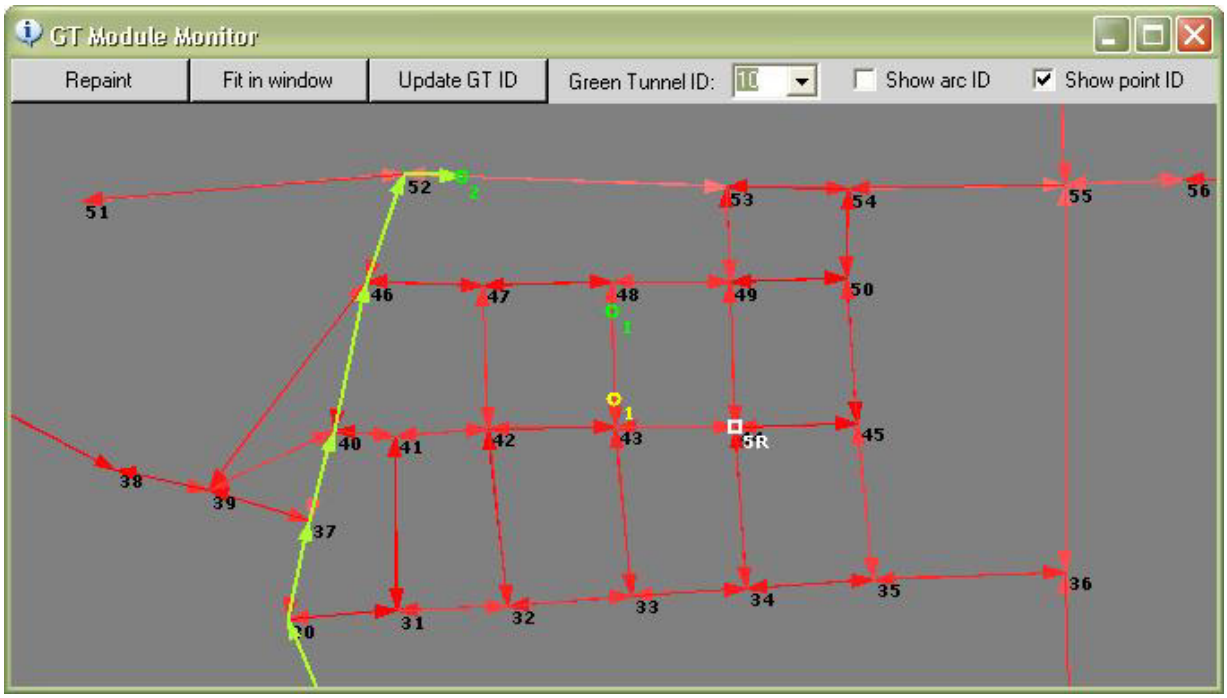


Figure 14 Green Tunnel Module Monitor main view

3.4.3 GPS Emulator (GPSemu)

As at the beginning of implementation phase we had no Vehicle Module, we decided to use a program called GPSemu (GPS Emulator) in order to be able to test other modules. The GPS emulator was developed by our friend. GPSemu enables testing the behavior of the GREEN TUNNEL SYSTEM when vehicles change their positions, as it generates signals similar to those generated by the real GPS system. The input parameters for the module are the route of the vehicle and its average speed. Then the GPS emulator generates proper signals and sends them using RS232 port.

3.5 Data encryption and modules' authorization

As mentioned above many times, all data transmitted in the GREEN TUNNEL SYSTEM is very crucial. Therefore it must be secured in a proper way. Also no unauthorized connection to any of the modules (e.g. Data Access Module or Vehicle Communication Module) should be acceptable. To ensure these requirements we use special communication classes, which use encryption and authorization methods. These classes will be described below.

Establishing an encrypted connection between any two modules is divided into two phases. The first phase is called the authorization phase, and the second – key exchange phase. The second phase comes directly after successful first phase. After both phases end with success, data transmission can be done.

In the process of authorization, the RSA algorithm is used. During the next phase, a key is generated in order to ensure secure data passing. All communication is enciphered with the use of Rijndael algorithm. A key for this algorithm is the key generated during the second phase. When the connection is closed, the key is automatically destroyed.

Generally all modules can be divided into two groups – servers and clients. Servers are these modules that wait for an incoming connection. Example of a server module can be Data Access Module. The second group consists of modules that initialize connections with the servers, e.g. Officer-on-Duty Module or Vehicle Module. There is also one module in the GREEN TUNNEL SYSTEM that is simultaneously a server and a client – a Vehicle Communication Module. Every of the servers and the Clients is delivered with the special set of passwords required in the process of authorization. These passwords are generated in the Administration & Monitoring Center, when a new Emergency Department or vehicle is going to be included into the GREEN TUNNEL SYSTEM. Then the keys should be delivered to Emergency Department (or vehicle) on a physical carrier, such as a diskette. A set for client module consists of a private key of RSA key pair, unique identifier ID, and an additional key K_0. A corresponding set of keys have to be provided for a server, which a client will connect with. So, the servers' key set consists of a public key if RSA key pair (the same pair as in the client), unique identifier ID (it is client's ID), an additional key K_0 (the same as the client one), and a public IP address of the client.

As many clients will connect with a specified server, the latter have to have corresponding set for each of them, as every client has its own set of keys.

The process of authorization goes as follow. When a client wants to connect to the server, it encrypts its ID using RSA algorithm with the private key, and sends it to the server. Server checks if the IP address of incoming connection is known and, if it is a registered IP address, decrypts incoming message. If decrypted data is equal with the ID that corresponds mentioned IP address, the server sends to the client enciphered message "OK" and the authorization phase is over. (Note: client should receive the "OK"-message and successfully decrypt it.) There is a problem with Vehicle Modules as they have no IP address. The solution is changing the IP address into a registration number of a car that specified Vehicle Module is installed in. If any of the above operations fail, the connection is broken and a client will not be able to communicate with the server.

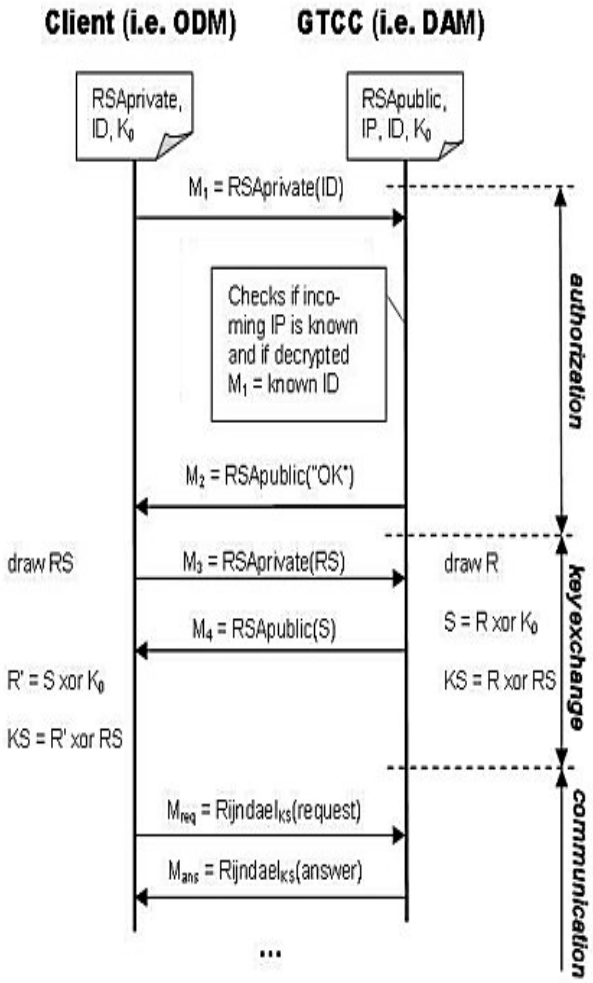


Figure 15 Establishing encrypted connection

During the key exchange phase, a key (for Rijndael algorithm) that will be used during data exchanging is established. The client draws an additional key RS (it has the same length as the K_0 key) and sends it to the server using RSA algorithm for encryption. The server draws an additional key R (the same as the corresponding K_0 key) and receives the RS from a client. Then it combines R with K_0 using a bitwise XOR operation and sends it to the client (using RSA

algorithm). A client receives this data and combines it with K_0 using bitwise XOR operation. At the end both sides, i.e. client and server, has to combine their key data with RS using bitwise XOR operation. The process of key generation is over, and the generated key has the same length as the K_0 key. Now secure data transmission can begin.

Both phases of establishing a connection are shown in Figure 15.

All keys used by the GREEN TUNNEL SYSTEM are of the specified length, but it is possible to change it in order to increase performance or fulfill exportation regulations of a given country. For details of security levels and key lengths see section 7.3.

4 Verification and testing

A system like GTS should be thoroughly tested before putting it into practice. As none of the local authorities would agree to test such a system in the real world, we were obliged to use software simulators to perform the tests. We used the Traffic Simulator (described in section 3.4.1) and Vissim (commercial traffic simulator available at Poznan University of Technology) for generating artificial traffic intensity. These systems work with a feedback loop – traffic simulators generate information about the traffic, which is processed by the GREEN TUNNEL SYSTEM. The system in turn generates steering signals for traffic lights controllers, which may affect the traffic and in consequence they must be taken into account in the next step of the traffic simulation. The Vissim simulator may implement almost any scheme of traffic lights controlling, however its use is limited to some five or six semi-complex crossroads. Our Traffic Simulator is a little simpler, but can simulate more crossroads. We used the GPSemu (section 3.4.2) for each vehicle's route simulation. This tool also worked with a feedback. The GREEN TUNNEL SYSTEM computed the route for the vehicle, and the GPSemu returns subsequent positions of the vehicle with requested frequency.

Having all that, we generated some accidents and obstacles in the city and observed the behavior of the system after setting some Green Tunnels. We compared the results with a plain simulation of the city, which involved only the traffic simulation part. In the majority of cases the arrival times were shorter with the use of our system.

Of course, these tests could not show any gain from using the Vehicle Modules. In order to test this part of the system, we invented another kind of tests. One of us, who knows the city quite well, drove a car. Another person generated requests in random places of the city. We measured how much time it took the driver to reach the place with and without the help of our system. While the time to reach well-known places did not differ much, the time of arrival at less popular places (e.g. detached house estates) decreased in most cases where the Vehicle Module was used to show the route.

Another question is the safety for every road user. As the system affects traffic lights controllers' work, there is theoretically a possibility of causing some dangerous situations resulting from wrong computation. We want to emphasize that such situations can never happen, as the traffic lights controlling is based on direction priorities. We remotely assign a higher priority to the direction on the route of the Green Tunnel, which causes automatic lengthening of the green light time by the controller. This is important for several reasons. First of all the controller maintains the table of colliding directions and takes care not to open the way for vehicles from such directions. Moreover, the traffic on the side directions (from the Green Tunnel's point of view) is not stopped entirely, for keeping on the traffic flow is one of the conditions to avoid traffic jams.

The above tests proved that our system is efficient and safe.

5 Summary

The whole idea of building the GREEN TUNNEL SYSTEM came to us while we were watching the traffic in Poznań and the problems that emergency vehicles faced when they needed to reach the place of an accident. When installed in the city the GREEN TUNNEL SYSTEM benefits in many ways. The officer-on-duty can prepare the request while receiving the call informing about the accident. The closest emergency vehicle can already start while the officer receives supplementary information from the caller. The officer need not select the vehicle manually, which reduces errors resulting from incomplete knowledge about the vehicle's position. The driver need not know the city plan as he sees the complete route on the screen. The misunderstandings resulting from the noise in the radio are eliminated. All this shortens the time of rescue vehicle arrival which results in lowering the possible losses.

The prototype we have built is simplified because its aim is to demonstrate the idea rather than provide ready-to-install equipment. The Vehicle Module computer can be integrated with the display to save the space on the dashboard. The RS232-to-OBD-II interface can be changed into another to support vehicles equipped with OBD-I connector or into some other sensors for older vehicles not equipped with an on-board computer. Appropriate type of wireless network can be designed for the given place of system installation.

The software part of the system can be improved in many ways. One of them is the possibility of unloading traffic jams, which can be added to the Crossroads Module. The information about the traffic intensity can be used to estimate how long is the queue of waiting cars and if there is any possibility to unload it by turning on certain green lights at the given crossroad, and probably also at the adjacent ones. However, the problem of unloading traffic jams is a very complex task, and algorithms have been proposed only for certain road networks [12][13][14].

The Green Tunnel System can be expanded on additional modules such as:

- module which enables arrangement of the best points of the police blockades, on the basis of the last known position and average speed of a criminal;

- module for cabs – the GREEN TUNNEL SYSTEM is able to calculate on demand the best route to a taxi passenger;
- module for regular drivers which provides them with information about current congestion in a city and shows the best route to a given destination.

The current GREEN TUNNEL SYSTEM implementation uses the .NET Framework, which is a new technology that enables building cross-platform applications. The GREEN TUNNEL SYSTEM is easy and cheap especially in big cities, where underlying infrastructure of traffic lights and communication network is well developed. Additional costs may appear if the traffic lights controllers do not enable remote parameter setting. However, the system can be also installed in cities where traffic lights controllers cannot be programmed remotely. In this case the system helps only assign vehicles to accidents and calculate the best routes.

6 References

[1] Mikolaj Sobczak, *Determining uncertain position of a ship*, Institute of Computer Science, Poznan University of Technology.

[2] Steven S. Skiena, *The Algorithm Design Manual*, Springer-Verlag Inc. New York, 1998

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, *Introduction to Algorithms.* The Polish edition, Wydawnictwo Naukowo-Techniczne Warszawa 2000.

[4] Traffic Lights Controller MSR 2002 – http://www.msrtraffic.com.pl

[5] RFM 433 Modules – http://www.rfm.com

[6] Magneto-Inductive Compass TCM 2-20 – http://www.tri-m.com

[7] Garmin III PLUS GPS receiver – http://www.garmin.pl

[8] Embedded PC – http://www.elmark.com

[9] LCD – http://www.sharp.com

[10] R.L. Rivest, A. Shamir, L. Adleman, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, (http://www.nist.gov)

[11] Joan Daemen, Vincent Rijmen: *AES Proposal: The Rijndael Block Cipher* (http://www.nist.gov)

[12] Penina Roberg, Christopher R. Abbess, *Diagnosis and treatment of congestion in central urban areas*, European Journal of Operational Research 104 (1998) pp 218–230

[13] Yen-Liang Chen, Hsu-Hao Yang, *Minimization of travel time and weighted number of stops in a traffic-light network*, European Journal of Operational Research 144 (2003) pp 565–580

[14] Penina Roberg, *The development and dispersal of area-wide traffic jams*, Report of the Department of Mathematics, Middlesex University

[15] T. J. McCabe, *A Complexity Measure*, IEEE Transactions on Software Engineering, vol. SE-2, Dec. 1976. 20

7 Appendices

7.1 Database description

The database stores all information processed by the system and servers also as a kind of shared memory, through which different modules working within The Green Tunnel Control Center can interchange data. Underline in the field name denotes a primary key. The complete entity diagram of the database is presented in Figure 7.

7.1.1 Basic Green Tunnel System tables

Arc

The table stores the information about all arcs in the city. By the term "arc" we understand a part of the street between two adjacent crossroads, which leads in one direction. Thus for a street which allows traffic in both directions we have two arcs. Contents of the table are generated from the city map by the Map Editor in the Administration Module. The description of each field in the table is given below.

Field	Type	Null	Default	Description
id_arc	int(10) unsigned		NULL	Unique identifier of the arc
id_in	int(10) unsigned		0	ID of the start point
id_out	int(10) unsigned		0	ID of the end point
Turn_back	int(10) unsigned	YES	NULL	ID of the arc in the opposite direction
lawful	tinyint(1)	YES	NULL	The possibility of turning back
cars	smallint(5) unsigned		0	Total traffic congestion
id_street	int(10) unsigned	YES	NULL	ID of the street that the arc is part of
lane	tinyint(3) unsigned		0	Number of lanes on one arc

number_start	smallint(5) unsigned	YES	NULL	House number at the beginning of the arc
number_end	smallint(5) unsigned	YES	NULL	House number at the end of the arc
id_zone	int(10) unsigned		0	ID of the zone the arc belongs to

Crossroads

The table stores information about all crossroads in the city. A crossroad is a set of points. These points are beginning or ending of arcs. Connections between these points (within a crossroad) define the structure of the crossroad. This structure is defined in Relation table. Contents of the Crossroads table are generated from the city map by the Map Editor in the Administration Module.

Field	Type	Null	Default	Description
id_crossroads	int(10) unsigned		NULL	Unique identifier of the crossroads
coord_x	int(11)	YES	NULL	The x coordinate
coord_y	int(11)	YES	NULL	The y coordinate
name	varchar(100)	YES	NULL	The description of the crossroads

GT (Green Tunnel)

The table stores the information about all green tunnels that are currently active. Contents of this and the next (GT_Entry) table are generated by the Green Tunnel Module. Each tuple in the table is a kind of list header, whereas the contents of the list are stored in the GT_Entry table in the form of a linked list. The starting and ending point values are copied from the user's request, and the identifier of the vehicle is the result of the Green Tunnel Module's resource allocation algorithm.

Field	Type	Null	Default	Description
id_gt	int(10) unsigned		NULL	Unique identifier of the Green Tunnel
id_start_arc	int(10) unsigned		0	ID of the first arc in the tunnel
id_start_shift	int(10) unsigned		0	Shift of the start point on the first arc
id_end_arc	int(10) unsigned		0	ID of the last arc in the tunnel
id_end_shift	int(10) unsigned		0	Shift of the end point on the last arc
id_gt_entry	int(10) unsigned		0	Pointer to the list of arcs in the gt_entry
id_vehicle	int(10) unsigned	YES	NULL	ID of the vehicle that follows the tunnel
id_next_gt	int(10) unsigned	YES	NULL	ID of the next Green Tunnel (applies only to tunnels set in response to PGT request)

GT Entry (Green Tunnel Entry)

This table, together with the GT table, stores complete information about green tunnels. Each tuple stores the reference to a certain point (from the Point table), and thus uniquely identifies all arcs, crossroads and relations on the vehicle's route.

Field	Type	Null	Default	Description
id_gt_entry	int(10) unsigned		NULL	Unique id of the entry
id_point	int(10) unsigned		0	ID of the point on the path of the tunnel
id_next_entry	int(10) unsigned	YES	NULL	ID of the next entry (NULL terminates the list)
id_gt	int(10) unsigned		0	ID of the Green Tunnel that the entry is part of
datetime	datetime	YES	NULL	Timestamp assigned by the CRM

Obstacle

Each tuple of this table stores information about one inaccessible arc in the city. The start field is mandatory and denotes the date and time when the arc became blocked. Usually it is the date and

time of adding the information by the officer. The end field is optional, as some obstacles may need to be in the system until manual removal. Contents of this table are produced by The Data Access Module, based on messages from The Officer-on-Duty modules.

Field	Type	Null	Default	Description
id_obstacle	int(10) unsigned		NULL	Unique ID of the obstacle
id_arc	int(10) unsigned		0	ID of the blocked arc
shift	int(10) unsigned		0	Offset of the obstacle on the arc
start	timestamp(14)		NULL	The date and time of the occurrence
end	timestamp(14)	YES	NULL	The estimated date and time of the removal
description	varchar(50)	YES	NULL	The description of the obstacle

Perm_tunnel (Permanent Green Tunnel)

This table stores information about all Permanent Green Tunnels entered by the officers. Contents of this table are written by The Data Access Module based on messages from The Officer-on-Duty Modules. The meaning of this and the next (Perm_entry) is analogous to the GT and GT_Entry tables. The Perm_tunnel table stores the header of the list, whereas the Perm_Entry table stores the contents organized on the form of linked list.

Field	Type	Null	Default	Description
id_perm_tunnel	int(10) unsigned		NULL	Unique id of the permanent green tunnel
id_user	int(10) unsigned		0	ID of the user who sent the request
id_start_entry	int(10) unsigned		0	ID of the first entry on the list
id_end_entry	int(10) unsigned		0	ID of the last entry on the list

Perm_entry (Permanent Green Tunnel Entry)

Detailed information about subsequent points on the route of the green tunnel is stored in this table. Each point of the tunnel is expressed as a pair [id_arc, shift], which uniquely identifies

certain place on the map. An optional timestamp can be associated with each point. This timestamp represents the latest point of arrival at the place.

Field	Type	Null	Default	Description
id_perm_entry	int(10) unsigned		NULL	Unique ID of the entry
id_arc	int(10) unsigned		0	ID of the arc that belong to the PGT
shift	int(10) unsigned		0	Shift of the point on the arc
datetime	datetime	YES	NULL	The he latest moment of arrival at the point
id_next_entry	int(10) unsigned	YES	NULL	ID of the next entry (NULL terminates the list)

Point

The table maintains the list of all beginnings and endings of arcs (connections between crossroads) and relations (connections within a crossroad).

Field	Type	Null	Default	Description
id_point	int(10) unsigned		NULL	Unique identifier of the point
coord_x	int(11)		0	The x coordinate of the point
coord_y	int(11)		0	The y coordinate of the point
id_crossroads	int(10) unsigned		0	ID of the crossroads

Relation

A relation is one possible movement within a crossroad, e.g. turning left. A relation is a connection between an arc ending in a crossroad and an arc starting in the crossroad. Thus, a set of relation defines the structure of a given crossroad.

Field	Type	Null	Default	Description
id_relation	int(10) unsigned		NULL	Unique id of the relation
id_in	int(10) unsigned		0	ID of the start point
id_out	int(10) unsigned		0	ID of the end point
lawful	tinyint(1)		0	The possibility of going through the relation

Request

The Request table stores all requests sent from The Officer-on-Duty Modules to The Data Access Module. The information in then used by The Green Tunnel Module for assigning individual vehicles to tasks and computing green tunnels.

Field	Type	Null	Default	Description
id_request	int(10) unsigned		NULL	Unique id of the request
id_user	int(10) unsigned		0	ID of the user who sent the request
id_vehicle_type	int(10) unsigned		0	ID of the requested vehicle type
id_station	int(10) unsigned	YES	NULL	ID of the station from which the request came
id_arc	int(10) unsigned		0	ID of the target arc
shift	int(10) unsigned		0	Shift on the arc
date_time	timestamp(14)	YES	NULL	The date and time of the occurrence
id_vehicle	int(10) unsigned	YES	NULL	ID of the vehicle assigned by the GTM
regions	tinyint(1)		0	The city is (1) or is not (0) divided into regions

Ret_tunnel (Return Green Tunnel)

This table stores the information about the vehicles' return tunnels. The information concerning the target of the tunnel and current vehicle's position is used to compute the green tunnel for the vehicle.

Field	Type	Null	Default	Description
id_ret_tunnel	int(10) unsigned		NULL	Unique id of the return tunnel
id_vehicle	int(10) unsigned		0	ID of the vehicle following the tunnel
id_arc	int(10) unsigned		0	ID of the target arc
shift	int(10) unsigned		0	Shift on the target arc
id_user	int(10) unsigned		0	ID of the user that sent the request
id_station	int(10) unsigned		0	ID of the station from which the request came

Role

Roles are certain sets of privileges defined in the system. Each user is assigned a role to be able to access the features he needs to do his job.

Field	Type	Null	Default	Description
id_role	int(10) unsigned		NULL	Unique id of the role
name	varchar(50)			Description of the role
ReqMgmt	Tinyint(1)	NO	0	Requests management privilege
PGTMgmt	Tinyint(1)	NO	0	Parmanent Green Tunnels management privilege
ObstMgmt	Tinyint(1)	NO	0	Obstacles management privilege
VehMgmt	Tinyint(1)	NO	0	Vehicles management privilege

UserMgmt	Tinyint(1)	NO	0	Users' accounts management privilege
MapEdit	Tinyint(1)	NO	0	Map edition privilege
LogView	Tinyint(1)	NO	0	Logs viewing privilege
StatView	Tinyint(1)	NO	0	Statistics viewing privilege

Station

A station is a place, where the Officer-on-Duty Module is installed (e.g. Health Care Station). Fields "ambulance", "police" and "fire" denote whether the station owns and can run vehicles of certain type (ambulances, police and fire rescue vehicles, respectively).

Field	Type	Null	Default	Description
id_station	int(10) unsigned		NULL	Unique id of the station
id_arc	int(10) unsigned			ID of the arc, where the station is located
shift	int(10) unsigned			Shift on the arc
name	varchar(50)			The name of the station
ambulance	tinyint(1)	YES	NULL	TRUE if it is an ambulance department
police	tinyint(1)	YES	NULL	TRUE if it is a police department
fire	tinyint(1)	YES	NULL	TRUE if it is a fire department
ip	varchar(15)			The IP address of the station

Street

The table stores the information about all streets in the city. A street is a set of arcs which have the same name on the city map.

Field	Type	Null	Default	Description
id_street	int(10) unsigned		NULL	Unique id of the street
name	varchar(50)			The name of the street

User

Each tuple in this table represents one user of the system. The login and password fields are used during the user's authorization process

Field	Type	Null	Default	Description
id_user	int(10) unsigned		NULL	Unique id of the user
id_role	int(10) unsigned		0	ID of the user's role
id_boss	int(10) unsigned	YES	NULL	ID of the user's boss
id_station	int(10) unsigned		0	ID of the station the user works in
login	varchar(50)			User's login
password	varchar(50)			User's password
name	varchar(50)			User's name
surname	varchar(50)			User's surname
phone	varchar(11)	YES	NULL	User's phone number

Vehicle

Each tuple in this table represents an individual vehicle. The registration field is used during the vehicle's authorization process. The pair [id_arc, shift] denotes current vehicle position. The position in updated by the Data Access Module based on the information obtained from the vehicle.

Field	Type	Null	Default	Description
id_vehicle	int(10) unsigned		NULL	Unique id of the vehicle
id_vehicle_type	int(10) unsigned		0	ID of the type of the vehicle
id_station	int(10) unsigned		0	ID of the station the vehicle belongs to
registration	varchar(10)			The vehicle's registration number

id_arc	int(10) unsigned		0	ID of the arc where the vehicle is at the moment
shift	int(10) unsigned		0	Shift on the arc
state	char(1)			The state of the vehicle (Available, Unavailable, Return tunnel, Less critical action)
id_sender	int(10) unsigned	YES	NULL	ID of the station which sent the request for the vehicle

Vehicle_type

This table stores the types of all available vehicles. The type of vehicle is assigned to a request instead of individual vehicle, to assure optimal allocation of resources.

Field	Type	Null	Default	Description
id_vehicle_type	int(10) unsigned		NULL	Unique id of the vehicle type
name	varchar(50)			The name of the type
symbol	varchar(10)	YES	NULL	Short symbol of the type
description	varchar(255)			Detailed description of the type

Zone

A zone is a set of streets, which are under control of one station. The division of the city into zones (or regions) is the implementation of the requirement that not all vehicles can go everywhere. For example accidents on some streets are serviced by one Health Center, while accidents on other streets are serviced by another Health Center.

Field	Type	Null	Default	Description
id_zone	int(10) unsigned		NULL	Unique id of the zone
name	varchar(50)	YES	NULL	The name of the zone
id_station	int(10) unsigned			The station which services the zone

7.1.2 GPS emulator tables

Format

This table stores information about different output data formats. The format value field may contain special tags recognized by the emulator.

Field	Type	Null	Default	Description
format_id	decimal(4,0)	YES	NULL	Unique id of the format
format_desc	varchar(255)	YES	NULL	The description of the format
format_value	varchar(255)	YES	NULL	The formatting string

GPS

The table stores information about all map files used in GPS Emulator together with the geographic coordinates of each part.

Field	Type	Null	Default	Description
lat	decimal(8,0)	YES	NULL	The latitude
lon	decimal(8,0)	YES	NULL	The longitude
filename	varchar(16)	YES	NULL	Map filename

Path

The table stores information about paths available for the GPS Emulator. The identifier of the path is equal to the identifier of the vehicle that follows certain path. The path is generated after The Green Tunnel Module calculates all green tunnels.

Field	Type	Null	Default	Description
path_id	decimal(8,0)		0	Unique id of the path
path_name	varchar(255)	YES	NULL	The name of the path

Path_item

This table actually stores all points on the vehicle's route. The path_id is equal to the identifier of the vehicle following the path. The item_id field is a zero-based index of each point on the route. The latitude and longitude are geographic coordinates, transformed to an integer.

Field	Type	Null	Default	Description
item_id	decimal(8,0)		0	The number of the item
item_lat	decimal(8,0)	YES	0	The point's latitude
item_lon	decimal(8,0)	YES	0	The point's longitude
path_id	decimal(8,0)		0	ID of the path

7.2 Database access classes

Class: CArc_MULTI

Tables: arc, point, street

Equivalent SQL query:

```
SELECT *
FROM ((arc
        LEFT JOIN point point1 ON arc.id_in=point1.id_street)
        LEFT JOIN point point2 ON arc.id_out=point2.id_point)
        LEFT JOIN street ON arc.id_street=street.id_street
WHERE id_street = ?;
```

Description: This class is designed to retrieve from the database information about all arcs (the number of lanes, traffic congestion etc.), together with the data about starting and ending points and with the street name. The id_street parameter is optional. If set, it reduces the output set to those arcs, which belong to the street with given identifier.

Class: CDBTraffic

Tables: arc

Equivalent SQL query:

```
SELECT id_arc, cars
FROM arc;
```

Description: This query retrieves information about recent traffic state. It is used for constructing the TrafficUpdate message.

Class: CGT

Tables: gt, arc, point

Equivalent SQL query:

```
SELECT id_gt, id_start_arc, id_start_shift, id_end_arc, id_end_shift, id_gt_entry,
        id_vehicle, a1s.coord_x, a1s.coord_y, a1e.coord_x, a1e.coord_y,
        a2s.coord_x, a2s.coord_y, a2e.coord_x, a2e.coord_y, r.id_station
FROM ((((((gt
        LEFT JOIN arc a1 ON gt.id_start_arc=a1.id_arc)
        LEFT JOIN arc a2 ON gt.id_end_arc=a2.id_arc)
        LEFT JOIN point a1s ON a1.id_in=a1s.id_point)
        LEFT JOIN point a1e ON a1.id_out=a1e.id_point)
        LEFT JOIN point a2s ON a2.id_in=a2s.id_point)
        LEFT JOIN point a2e ON a2.id_out=a2e.id_point)
        LEFT JOIN request r ON gt.id_vehicle=r.id_vehicle;
```

Description: This query works in pair with the next in order to obtain the information about all active green tunnels. Results of this query give some basic information about the green tunnel (including the identifier of the first arc) and provide a pointer for the next entry.

Class: CGTEntry

Tables: gt, gt_entry, arc

Equivalent SQL query:

```
SELECT s.id_gt, arc.id_arc,
FROM (gt_entry s
        LEFT JOIN gt_entry e ON s.id_next_entry=e.id_gt_entry)
        LEFT JOIN arc ON (arc.id_in = s.id_point AND arc.id_out = e.id_point)
WHERE id_arc IS NOT NULL;
```

Description: Results of this query return the list of arcs belonging to each of the tunnels. Joins are necessary as the table gt stores the identifiers of the points. Arcs belonging to one green tunnel are marked with the same gt_id.

Class: CGTPoints

Tables: gt_entry, point

Equivalent SQL query:

```
SELECT gt_entry.id_gt_entry, gt_entry.id_point, gt_entry.id_next_entry,
    gt_entry.id_gt, point.coord_x, point.coord_y
FROM gt_entry
    LEFT JOIN point ON gt_entry.id_point=point.id_point;
```

Description: This query returns the coordinates of points belonging to a green tunnel.

Class: CLastId

Tables: *none*

Equivalent SQL query:

```
SELECT last_insert_id();
```

Description: The query returns the id of lately inserted obstacle.

Class: CLoginAckRecordset

Tables: user, station

Equivalent SQL query:

```
SELECT *
FROM (user
    LEFT JOIN station ON user.id_station=station.id_station)
    LEFT JOIN role ON user.id_role=role.id_role
WHERE user.login = ?;
```

Description: The recordset is used to retrieve from the database the information about a user who is trying to authenticate in the system. The information is used to verify his password. If the authorization process succeeds, all information send to the Officer-on-Duty module.

Class: CObstacle

Tables: obstacle

Equivalent SQL query:

```
SELECT id_obstacle, id_arc, start, end, description
FROM obstacle;
```

Description: This query returns the information about all obstacles in the city.

Class: CPGT_MULTI

Tables: perm_tunnel, perm_entry, arc, street, user

Equivalent SQL query:

```
SELECT t.id_perm_tunnel, arc.id_arc, street.name
FROM (((perm_tunnel t
    LEFT JOIN perm_entry e ON t.id_start_entry = e.id_perm_entry)
    LEFT JOIN arc ON e.id_arc = arc.id_arc)
    LEFT JOIN street ON arc.id_street = street.id_street)
    LEFT JOIN user ON t.id_user = user.id_user
WHERE id_station = ?;
```

Description: This recordset contains information about Permanent Green Tunnels. The resulting set may be narrowed to only those, which were sent from the station of given identifier.

Class: CRequest_MULTI

Tables: request, arc, street, user, vehicle_type

Equivalent SQL query:

```
SELECT arc.id_arc, arc.id_in, arc.id_out, r.shift, r.date_time, street.name, user.name
    user.surname, user.phone, vt.name, vt.symbol
FROM (((request r
    LEFT JOIN arc ON r.id_arc=arc.id_arc)
    LEFT JOIN street ON arc.id_street=street.id_street)
```

```
        LEFT JOIN user ON r.id_user=user.id_user)
        LEFT JOIN vehicle_type vt ON r.id_vehicle_type=vt.id_vehicle_type
    WHERE r.id_station = ?;
```

Description: This query is used to obtain complete information about all requests, including the target location, the name of the user, who prepared the request. Optional parameter (id_station) allows to select only the requests from the station of given identifier.

Class: CSender

Tables: request

Equivalent SQL query:

```
SELECT id_station, id_request
FROM request
WHERE id_vehicle=?;
```

Description: Finds the identifier of the sender station for the vehicle of a given ID.

Class: CVehicle

Tables: vehicle

Equivalent SQL query:

```
SELECT id_vehicle, id_vehicle_type, id_station, id_arc, shift, state, registration
FROM vehicle
WHERE id_station = ?;
```

Description: Returns the information about all vehicles available in the city, including the state and position of each vehicle.

Class: CVehicle_type

Tables: vehicle_type

Equivalent SQL query:

```
SELECT id_vehicle_type, name, symbol, description
FROM vehicle_type;
```

Description: Returns names, symbols and descriptions of the types of rescue vehicles.

Class: CVehicleA

Tables: vehicle

Equivalent SQL query:

```
SELECT id_vehicle
FROM vehicle
WHERE state="A";
```

Description: Returns the collection of vehicles which are active at the moment.

Class: CVehicleCount

Tables: vehicle

Equivalent SQL query:

```
SELECT Count(id_vehicle)
FROM vehicle
WHERE id_vehicle_type = ? AND state = ?;
```

Description: The query designed to return the count of vehicles of given type in a given state (mostly – vehicles which are available).

Class: CVehicleLogin

Tables: vehicle, vehicle_type

Equivalent SQL query:

```
SELECT v.id_vehicle, v.id_vehicle_type, v.id_station, v.registration, v.id_arc,
        v.shift, v.state, vt.name, vt.symbol, vt.description
FROM vehicle v
```

```
        LEFT JOIN vehicle_type vt ON v.id_vehicle_type=vt.id_vehicle_type
    WHERE v.registration = ?;
```

Description: Returns the information concerning the vehicle of a given registration number. A query designed to retrieve information needed to authenticate a Vehicle Module in the system.

Class: CVehiclePos

Tables: arc, point, vehicle

Equivalent SQL query:

```
SELECT p.coord_x, p.coord_y, k.coord_x, k.coord_y, v.shift
FROM vehicle v
    LEFT JOIN arc ON v.id_arc=arc.id_arc
    LEFT JOIN point p ON arc.id_in=p.id_point
    LEFT JOIN point k ON arc.id_out=k.id_point
WHERE v.id_vehicle=?
```

Description: Returns the position of a vehicle as the coordinates of an arc and the offset from the beginning of the arc.

Class: CVehicleState

Tables: vehicle

Equivalent SQL query:

```
SELECT state
FROM vehicle
WHERE id_vehicle=?
```

Description: Returns current state of the vehicle of a given identifier.

7.3 Security features

7.3.1 Security level and key sizes

When generating passwords and keys for the modules, it is possible to choose one of the three security levels: LOW, MEDIUM, and HIGH. Each of them characterizes with different key sizes, what entails ciphering performance. The security levels are made for adjusting the safety of the system, but in order to fulfill exportation regulations of a given country as well. Security levels and corresponding key sizes are presented in the table below.

Security level name	RSA key size	K0 key size	Rijndael key size
LOW	384 bits	128 bits	128 bits
MEDIUM	576 bits	192 bits	192 bits
HIGH	768 bits	256 bits	256 bits

7.3.2 Password file structure

Passwords for modules are kept in specially designed files. Structure and names of these files are shown below.

a) file with keys for client – file name: *csidec.dat*

Parameter	Size in bytes	Desrciption
FILE_TYPE	4	Determines file type: 2 – client of non-vehicle module, 3 – client of vehicle module
ED/MUV_TYPE	4	Determines type of a client: 0 – non-vehicle client, 1 – vehicle client
SEC_LEVEL	4	Determines the level of security: 1 – low, 2 – medium, 3 – high
RSA_PRIVATE_KEY	219–435	RSA private key structure. Length depends on the chosen security level. For details see below.
ID	16	Holds the unique ID for the module.
K_0	16-32	Holds additional K_0 key, which length depends on the chosen security level.

b) file with keys for server – file name: *ssides.dat*

Parameter	Size in bytes	Desrciption
FILE_TYPE	4	Determines file type: 0 – server for non-vehicle modules, 1 – server for vehicle modules
ED_COUNT	4	Determines the number of known Emergency Department or Vehicle Modules
ITEM_1	95–159	Hold information for each known Emergency Department or Vehicle Module. For details see below.
ITEM_2	95–159	
…	…	
ITEM_k	95–159	

Structure of *ITEM_x*

Parameter	Size in bytes	Desrciption
ED/MUV_TYPE	4	Determines type of a client: 0 – non-vehicle client, 1 – vehicle client
IP_ADDRESS/ REGISTRATION_NUMBER	8	Determines the IP address of known Emergency Department or Registration Number of car in which Vehicle Module is installed
SEC_LEVEL	4	Determines the level of security: 1 – low, 2 – medium, 3 – high
RSA_PUBLIC_KEY	51–99	RSA public key structure. Length depends on the chosen security level. For details see below
ID	16	Holds the unique ID for the module.
K_0	16-32	Holds additional K_0 key, which length depends on the chosen security level.

7.3.3 RSA public and RSA private key structure

RSA key consists of a several parameters, which are: *D*, *DP*, *DQ*, *Exponent*, *InverseQ*, *Modulus*, *P*, and *Q*. All these parameters creates the RSA private key, while the public key is created with the use of *Modulus* and *Exponent*.

Enumerated parameters have different lengths, dependant on the security level. Relationship between security levels and RSA parameters is shown below.

Parameter	Security level (sizes in bytes)		
	HIGH	MEDIUM	LOW
D	96	72	48
DP	48	36	24
DQ	48	36	24
Exponent	3	3	3
InverseQ	48	36	24
Modulus	96	72	48
P	48	36	24
Q	48	36	24

7.4 GPS Receiver Characteristics and Protocol

7.4.1 Receiver characteristics

The GPS device used in the sample application is Garmin 35 PC GPS receiver. Its description and protocol specification is presented below.

Receiver features:

- Cold start: 45 s.
- Warm start: 15 s.
- Refresh interval: 1 sec. – 15 min.
- Interface: two-channel RS-232.
- Protocol: NMEA 0183 version 2.0 ASCII.
- Connection: power cord, data cord.

Physical characteristics:

- Dimensions: 56.4mm x 96.3mm x 26.7mm
- Weight: 110 g (not including cords).
- Operation temperature: -30°C to + 80°C.
- Power supply: 6-40 V from an external source.
- Emergency supply: 3V lithium battery.
- Antenna: built-in.

7.4.2 Communication protocol

An NMEA standard defines an electrical interface and data protocol for communications between marine instrumentation. (They may also have standards for other things.) These standards allow a single “talker”, and several “listeners” on one circuit.

Under the NMEA-0183 standard, all characters used are printable ASCII text (plus carriage return and line feed). NMEA-0183 data is sent at 4800 baud.

The data is transmitted in the form of “sentences”. Each sentence starts with a “$”, a two letter “talker ID”, a three letter “sentence ID”, followed by a number of data fields separated by commas, and terminated by an optional checksum, and a carriage return/line feed. A sentence may contain up to 82 characters including the “$” and CR/LF.

If data for a field is not available, the field is simply omitted, but the commas that would delimit it are still sent, with no space between them.

Since some fields are variable width, or may be omitted as above, the receiver should locate desired data fields by counting commas, rather than by character position within the sentence.

The optional checksum field consists of a “*” and two hex digits representing the exclusive OR of all characters between, but not including, the “$” and “*”. A checksum is required on some 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.

Some common talker IDs are: GP Global Positioning System receiver LC Loran-C receiver OM Omega Navigation receiver II Integrated Instrumentation

A talker typically sends a group of sentences at intervals determined by the unit's update rate, but generally not more often than once per second.

7.4.3 Hardware connection

The recommended interconnect wiring is a shielded twisted pair, with the shield grounded only at the talker. The standards do not specify the use of any particular connector.

The NMEA-0180 and 0182 standards say that the talker output may be RS-232, or from a TTL buffer, capable of delivering 10 mA at 4 V. A sample circuit shows an open collector TTL buffer with a 680 ohm resistor to +12 V, and a diode to prevent the output voltage from rising above

+5.7 V. NMEA-0183 accepts this, but recommends that the talker output comply with EIA-422. This is a differential system, having two signal lines, A and B. The voltages on the "A" line correspond to those on the older TTL single wire, while the "B" voltages are reversed (while "A" is at +5, "B" is at ground and vice versa). In either case, the recommended receive circuit uses an opto-isolator with suitable protection circuitry. The input should be isolated from the receiver's ground. In practice, the single wire, or the EIA-422 "A" wire may be directly connected to a computer's RS-232 input.

7.4.4 NMEA settings for the RS232

Baudrate	4800
Data bits	8 (Bit 7 set to 0)
Stop bits	1 or 2
Parity	none
Handshake	none

7.4.5 Sample Sentence

```
$GPRMC,225446,A,4916.45,N,12311.12,W,000.5,054.7,191194,020.3,E*
68<CRLF>
```

Field value	Description	Value
$	Sentence start	
GP	Talker ID	Global Positioning System receiver
RMC	Sentence ID	Recommended minimum specific GPS/Transit data
225446	Time of format HHMMSS UTC	22:54:46 UTC

A	Receiver warning A = OK, V = warning	OK
4916.45,N	Latitude	49 deg. 16.45 min North
12311.12,W	Longitude	123 deg. 11.12 min West
000.5	Speed over ground, Knots	0.5
054.7	Course Made Good, True	
191194	Date of format DDMMYY	19 November 1994
020.3,E	Magnetic variation	20.3 deg East
*68	Mandatory checksum	
<CRLF>	Frame end	

7.5 Magnetoinductive compass characteristics

7.5.1 Compass characteristics

The TCM2 is a low-power, high-performance Electronic Compass Sensor Module. The TCM2 combines a three-axis magnetometer and a high-performance two-axis tilt sensor in a small package.

Tilt	up to 20°
Accuracy at the ground level	0.5° RMS (in any environment)
Accuracy with a tilt	1° RMS (in any evironment
Magnetometer	three-axis
Tilt sensor	two-axis
Technology	magneto-inductive

7.5.2 Compass communication protocol

Operating mode

The compass can operate in two modes, as described below:

- standby mode

  During Standby mode the TCM2 is idle and not sampling any sensors. It is possible to configure and verify the TCM2's user parameters in the Standby mode (set sampling rate, filter parameters, etc.) It is also possible to query the TCM2 for single updates of compass heading, pitch and roll, magnetic field strength, and temperature. Because the TCM2's sensors are not sampling continuously during Standby, filtering is automatically disabled, and power consumption is reduced.

- continuous sampling mode

  In this mode, the TCM2 is actively collecting sensor data at the prescribed sampling rate. The principle means of output from the TCM2 is via RS232 serial communications. The

TCM2 may be configured either to output the latest sensor data on a continuous, unprompted basis, or to provide an update only upon demand.

Compass input – command syntax

There are three types of commands that may be issued to the TCM2:

- commands to set user-definable parameters have the following syntax:

  `<parameter>=<value><cr>`

- commands to query the TCM2 for data or for the stored value of user-definable parameters with the following syntax:

  `<parameter>?<cr>`

- action commands with the syntax:

  `<command><cr>`

All commands must be followed with a <cr>, or <cr><lf>. The <lf> characters are ignored by the TCM2, but are supported to allow compatibility with a variety of terminals.

Compass output – response syntax and error codes

The TCM2 will transmit data across the RS232 interface in response to input commands, and will also transmit data output words automatically when placed in continuous output mode. This data output word may be configured for the desired format and configuration. It is possible to select either NMEA 0183, or TCM2 standard output word formats. The response to the various commands is as follows:

Input Command	TCM2 Response
Valid parameter-setting commands	:<cr><lf>
Valid action command	varies according to command
Valid parameter query commands	:<parameter>=<value><cr><lf>
Invalid, or unrecognized command	E<code><cr><lf>
Valid sensor query command	varies according to command

The ":" character signifies a successfully identified and executed command.

We decided to use the TCM2 standard, as it is capable of delivering more information. The TCM2 standard output format may be configured to provide all of the sensor data parameters available, or only some required parameters, depending on the configuration. The word format is:

```
$C<compass>P<pitch>R<roll>X<Bx>Y<By>Z<Bz>T<temp>E<error code>*checksum<cr><lf>
```

Error codes are transmitted by the TCM2 during responses to commands, or as part of TCM2 messages in continuous output mode. Error codes are given by three ASCII characters representing hexadecimal digits Each error condition corresponds to one bit within one of the hexadecimal digits. When the error condition exists, that bit will be set equal to 1 in the error code transmitted by the TCM2. The error conditions and their corresponding bit locations is given below:

	Bit	Error condition
First digit	3 (MSB)	EEPROM1 error
	2	EEPROM2 error
	1	Reserved (always 0)
	0 (LSB)	Reserved (always 0)
Second digit	3 (MSB)	Reserved (always 0)
	2	Command parameter invalid
	1	Reserved (always 0)
	0 (LSB)	Command invalid or not available on current model of TCM2
Third digit	3 (MSB)	Reserved (always 0)
	2	Magnetometer out of range
	1	Inclinometer out of range
	0 (LSB)	Magnetic distortion alarm

7.5.3 Commands used in the Vehicle Module application

Command: Enter continuous sampling mode

Usage: go<cr>

Description: Instructs TCM2 to enter Continuous Sampling mode from the Standby mode. The TCM2 will immediately begin sampling sensors at the rate specified (by the "clock" command). The TCM2 will automatically transmit data at the sampling rate, and according to the output word format currently specified. To exit the TCM2 and return to Standby mode, issue the Halt Continuous Sampling ("h") command.

Command: Halt continuous sampling, enter standby mode

Usage: h<cr>

Description: Instructs TCM2 to exit Continuous Sampling mode and enter the Standby mode. If this command is received while the TCM2 is transmitting an output word, the remainder of the output word will be sent before the TCM2 changes modes.

Command: Set Clock Rate

Usage: clock=nn.nn<cr>

Valid Values: From 5 Hz to 40 Hz

Description: Enables the user to have fine control of the clock rate. The ability to sample at the higher speeds must be matched with the amount of data being output at each sample. At 9600 baud, each character requires about 1 millisecond to output, so at 30 Hz, at most 30 characters can be output which is not enough for all data outputs to be enabled in the output word.

Command: Set compass units

Usage: uc=n<cr>

Valid Values: "d" – degrees (360° in a full circle); "m" – mils (6400 mils in a full circle)

Description: Sets the units to be used for input/output of heading data.

Command: Set RS232 output word format

Usage: sdo=n<cr>

Valid Values: "t" – standard TCM2 output word; "n" – NMEA output format; "m" – mouse output format

Description: Sets the output word format to be used in response to the Single update output word command and in continuous output mode.

Command: Enable compass data for output word

Usage: ec=n<cr>

Valid Values: "e" – compass data enabled (will be included in output word); "d" – compass data disabled (will be excluded from output word)

Description: Instructs the TCM2 to either enable or disable compass data for inclusion in the TCM2 standard output word.

7.5.4 Hardware connection

The TCM2 utilizes a standard straight-post, polarized 10-pin header for power and data connections. The header is MOLX 22-03-2101style, with 0.100" centers. The mating receptacle is MOLX 22-01-3107, or equivalent. The crimps are MOLX 08-50-0114. The pin-out for this connector is diagrammed below.

Pin NO	In / out	Description
1	In	Voltage supply (5V, +/- 5%)
2	In	Voltage supply (6-25 V unregulated)
3	-	Power ground
4	In	RxD (RS 232)

5	Out	TxD (RS 232)
6	Out	Mouse output
7	-	Data ground
8	Out	Analog output 2
9	Out	Analog output 1
10	-	Data ground

Connection details:

Analog output is available on all TCM2 models. Mouse mode is for Virtual Reality applications. Supply voltage may either be supplied as 6-25V unregulated, or 5V regulated (±5%). (Power must not be applied to both power pins simultaneously.) The Data Ground, pin 7 of the connector, should be connected to RS232 ground. The RxD receive pin, pin 4, accepts voltages from -15V to 15V (-5V to 5V minimum). The TxD transmit pin outputs -5V to 5V. These ranges are compatible with most RS232 transceiver chips.

7.5.5 Settings for RS 232 communication

The serial communications interface is RS232, with the following parameters:

Baudrate	Configurable (300 – 38 400)
Data bits	8
Stop bits	1
Parity	none
Handshake	none

7.5.6 Sample output word

This sample assumes, that the device is configured to output the compass reading and the temperature only.

```
$C328.3T22.3*checksum<cr><lf>
```

This gives the information about the compass heading (328.3°) and the temperature (22.3 °C).

7.6 The OBD-II – RS 232 interface characteristics

7.6.1 Circuit description

The OBD (On Board Diagnostics) is a port for the connection of test equipment to the vehicle's on-board computer. Data is transferred serially between the vehicle and the external equipment using this connector in a manner specified by the Society of Automotive Engineers (SAE). The connections are actually serial links, but as the voltage levels and the protocols differ, personal computers' serial ports (i.e. RS 232) cannot be directly used to communicate with vehicles. The OBD-II comes in three standards: J1850 (PWM), J1850 (VPW) and ISO-9141 (a.k.a. KWP14230). The designed interface operates only with ISO-9141-compatible vehicles, as this standard is most popular among European vehicles.

The ELM323 is a 14-pin integrated circuit that, with only a few external components, is able to interface to the OBD port, interpreting the data signals and reforming them as standard ASCII characters. This allows communicating with the vehicle using only a standard serial port and communication software.

7.6.2 Interface schematics

The circuit is the typical ELM 323 application, and requires several external components to work. Circuit power is obtained from the vehicle (via OBD pins 16 and 5) and after some minor filtering is presented to the 5V voltage regulator. The regulator powers several points in the circuit as well as the LEDs. The remaining two connections to the vehicle (OBD pins 7 and 15) are for two data lines prescribed by the ISO-9141 standard. The ELM 323 controls both lines through the NPN transistors with the pullup resistors connected to their collectors. Data is received from the K line of the OBD bus and inverted by the PNP transistor. This transistor raises the threshold voltage to about 4V from the inherent 2.5V with the CMOS input of the ELM 323.

A very basic RS232 interface is connected to pins 5 and 6 of the chip. This circuit "steals" the power from the host computer in order to provide a full swing of the RS232 voltages without the need for a negative supply. RS232 data from the computer is directly connected to pin 6 of the IC

through the 47kΩ current limiting resistor. Transmission to the computer is via the PNP transistor connected to pin 6. This transistor allows the output voltage to swing between +5V and the negative voltage stored on the 0.1 µF capacitor (which is charged by the computer's TxD line). Although simple, this interface is quite effective for this application.

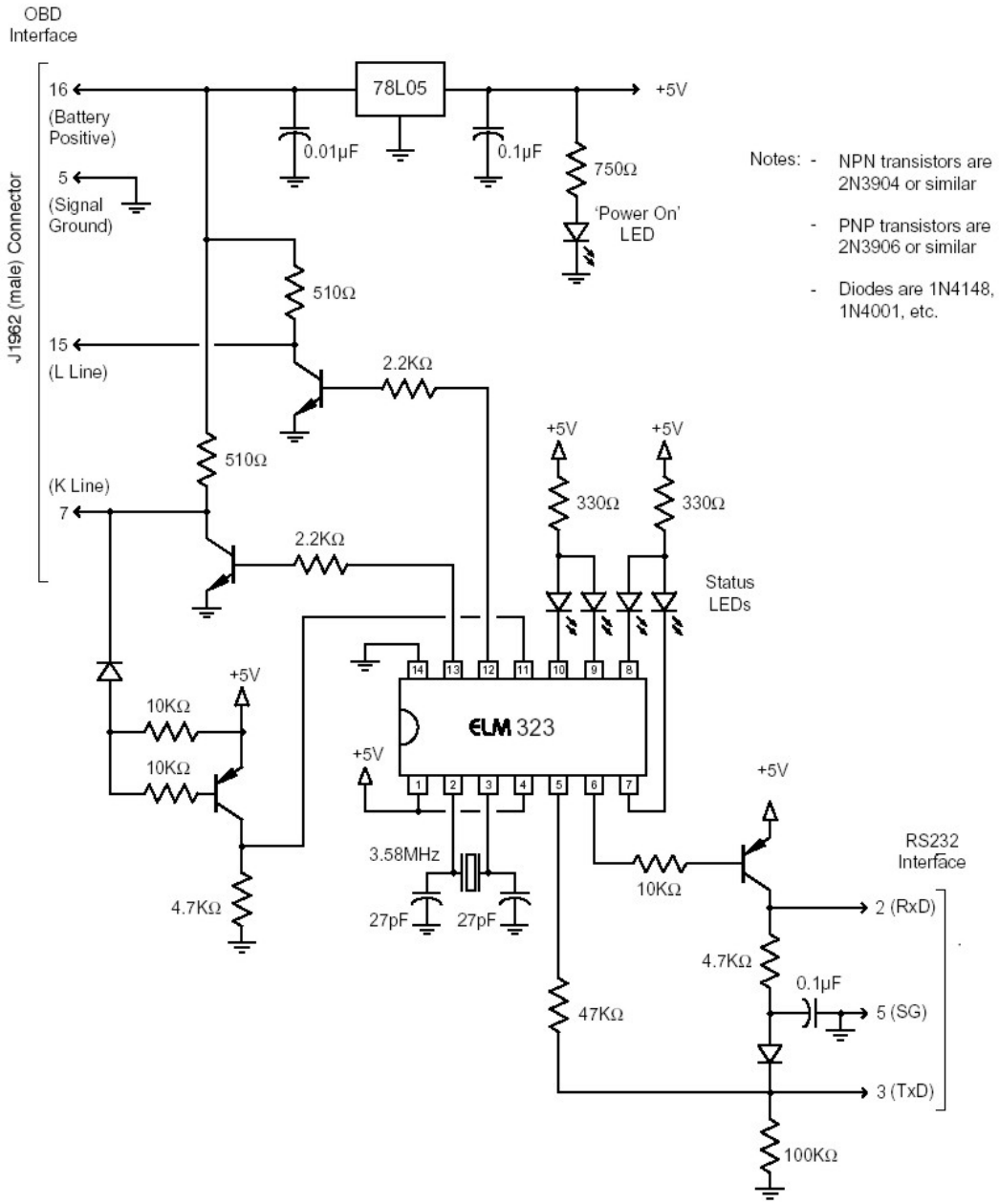


Figure 16 The OBD-II - RS232 interface schematic

7.6.3 Connection pinouts

RS – 232 connection

(solder side of the DB-9 female connector)

Pin NO	Description
2	Data transferred to the PC
3	Data received from the PC
5	Ground connection

OBD-II connection

(J1962 male connector)

Pin NO	Description
5	Signal ground
7	K Line
15	L Line
16	Battery positive

7.6.4 OBD protocol description

Before any communication can take place, the OBD bus must be first initialized. This task is done by the ELM 323 chip before the first communication and requires no interaction from the user.

The interface automatically distinguishes between commands for the vehicle and for the chip. If bytes received in the RS 232 bus do not begin with the letters “A” and “T”, they are assumed to be commands for the vehicle’s OBD bus. The bytes will be tested to ensure that they are valid pairs of hexadecimal digits, and will be combined into bytes for transmitting. No checks are made as to the validity of the OBD command.

OBD commands are actually sent to the vehicle’s computer in a data packet. The standard requires that three header bytes and an error checksum be included with every message. The interface inserts these automatically.

The SAE standards specify that each group of bytes sent to the vehicle must adhere to a set format. The first byte (known as the “mode”) always describes the type of data being requested, while the next bytes specify the actual information required (this is called the “parameter identification” or PID number). The modes and PIDs are described in detail in the SAE document

J1979 and may be expanded on by the manufacturers. The nine modes described in the document are:

Mode NO	Description
01	Show current data
02	Show freeze frame data
03	Show diagnostic trouble codes
04	Clear trouble codes and stored values
05	Test results, oxygen sensors
06	Test results, non-continuously monitored
07	Test results, continuously monitored
08	Special control mode
09	Request vehicle information

Within each mode, PID 00 is normally reserved to show which PIDs are supported by that mode. Mode 01, PID 00 must be supported by all vehicles. After issuing at the prompt:

0	1	2	3	4	5	6
>	0	1		0	0	\n

(“>” is the prompt character, blank cell denotes a space), the vehicle responds with the message, which may be as follows:

0	1	2	3	4	5	6	7	8	9	10
4	1		0	0		B	E		1	F
11	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	
	B	8		1	0		\r	\n	>	

The 41 00 signifies a response (first bit set $1000_{(b)}=4_{(d)}$) from a mode 1, request from PID 00. The next four bytes (BE 1F B8 10) represent the requested data – in this case a bit pattern showing the PIDs supported by this vehicle and this mode (1-supported, 0-not supported). The detailed information concerning all modes and PIDs may be obtained from SAE. Commands used in the Vehicle Module application will be discussed in the next section.

7.6.5 Commands used in the Vehicle Module application

Commands used to control the ELM323 chip

The ELM323 chip's factory default settings allow it to operate without any further configuration, however it is a good practice to make sure that the echoing of the characters received from the PC is turned off, which simplifies the structure of response parser. For this reason the ATE command is used as follows:

0	1	2	3	4
A	T	E	0	\n

Commands used to query the vehicle's status

The most important thing that is needed to compute the position is the vehicle's speed. Other data is of less importance for the Vehicle Module application, thus it won't be queried.

The command used to obtain the vehicle's speed is the "0D" command in mode "01". The query has the following form:

0	1	2	3	4
0	1	0	D	\n

The response from the interface is (blank cells denote a space – ASCII 32):

0	1	2	3	4	5	6	7	8	9	10	11
4	1		0	D		V1	V2		\r	\n	>

Where V1 and V2 denote the high order and low order nibbles respectively. Suppose V1="4" and V2="C", what gives the speed of 4*16+12=76 kph. The ">" character is the interface's prompt.

7.6.6 Settings for RS 232 communication

The RS 232 settings on the interface's side are hard-configured to the following:

Baudrate	9600
Data bits	8
Stop bits	1
Parity	none
Handshake	none

All responses from the interface are terminated with a single carriage return character and a single line feed character (the <cr><lf> sequence).

7.7 Serial port component

7.7.1 Component design & implementation

As several devices are connected to a serial port, we needed to develop a component, which can talk to this port. One of the requirements was to use only the WinAPI functions. Another requirement for the class is to service different devices. The GPS receiver and the RFM receiver are different data sources. The GPS receiver provides data regularly in known intervals, whereas transmissions through the RFM receiver happen in apriori unknown moments of time (when the Vehicle Module or the Vehicle Communication Module requests a communication). To provide universal and effective mechanism of communication the port is serviced in an overlapped-mode. The overlapped operation allows calling ReadFile and WriteFile asynchronously. The completion of the operation is signaled as a port event. Moreover, some operations (like reading and writing) can be performed simultaneously with one file handle. As a consequence of this choice, the component uses two separate threads, which service writing the data to the port and listening for incoming data and port events. The class provides buffers for both incoming and outgoing data. Care is taken about synchronization as these buffers are used in different threads (a single-producer multi-consumer scheme). Also simple mechanism is implemented to allow the support for different data formats and frames. The general CCustomPort class has several virtual methods, which can be overridden in descendants of this class to decode incoming data in different format or to decode outgoing data. The descendant class can use its own buffers and synchronization objects. The basic class provides methods to send and receive data in 1-byte packets.

We have used the standard Windows API functions to communicate over the serial port. The functions used strictly for port operation are listed below

- CreateFile - opens the serial port
- ReadFile - reads the data from the serial port
- WriteFile - sends the data over the serial port

- GetCommState - fills in a device-control block (a DCB structure) with the current control settings for a serial port
- SetCommState - configures a port according to the specifications in a device-control block (a DCB structure).
- GetCommMask - retrieves the value of the event mask for a serial port
- SetCommMask - specifies a set of events to be monitored for a serial port
- GetCommTimeouts - retrieves the time-out parameters for all read and write operations on a port
- SetCommTimeouts - sets the time-out parameters for all read and write operations on a specified port
- PurgeComm - discards all characters from the output or input buffer of a specified port. It can also terminate pending read or write operations on the resource.

Additional functions were necessary to service events in overlapped operation mode:

- CreateEvent - creates a named or unnamed event object
- SetEvent - sets the state of the specified event object to signaled
- ResetEvent - sets the state of the specified event object to nonsignaled
- WaitCommEvent - waits for an event to occur for a specified port
- WaitForMultipleObjects – waits for any of the events to change into signaled state or until a timeout elapses

As a synchronization object, the CRITICAL_SECTION structure was used with the following operations:

- InitializeCriticalSection - initializes a critical section object
- EnterCriticalSection - waits for ownership of the specified critical section object
- LeaveCriticalSection - releases ownership of the specified critical section object

- DeleteCriticalSection - releases all resources used by an unowned critical section object

Port operations (read, write and status check) are presented on the sequence diagram below.

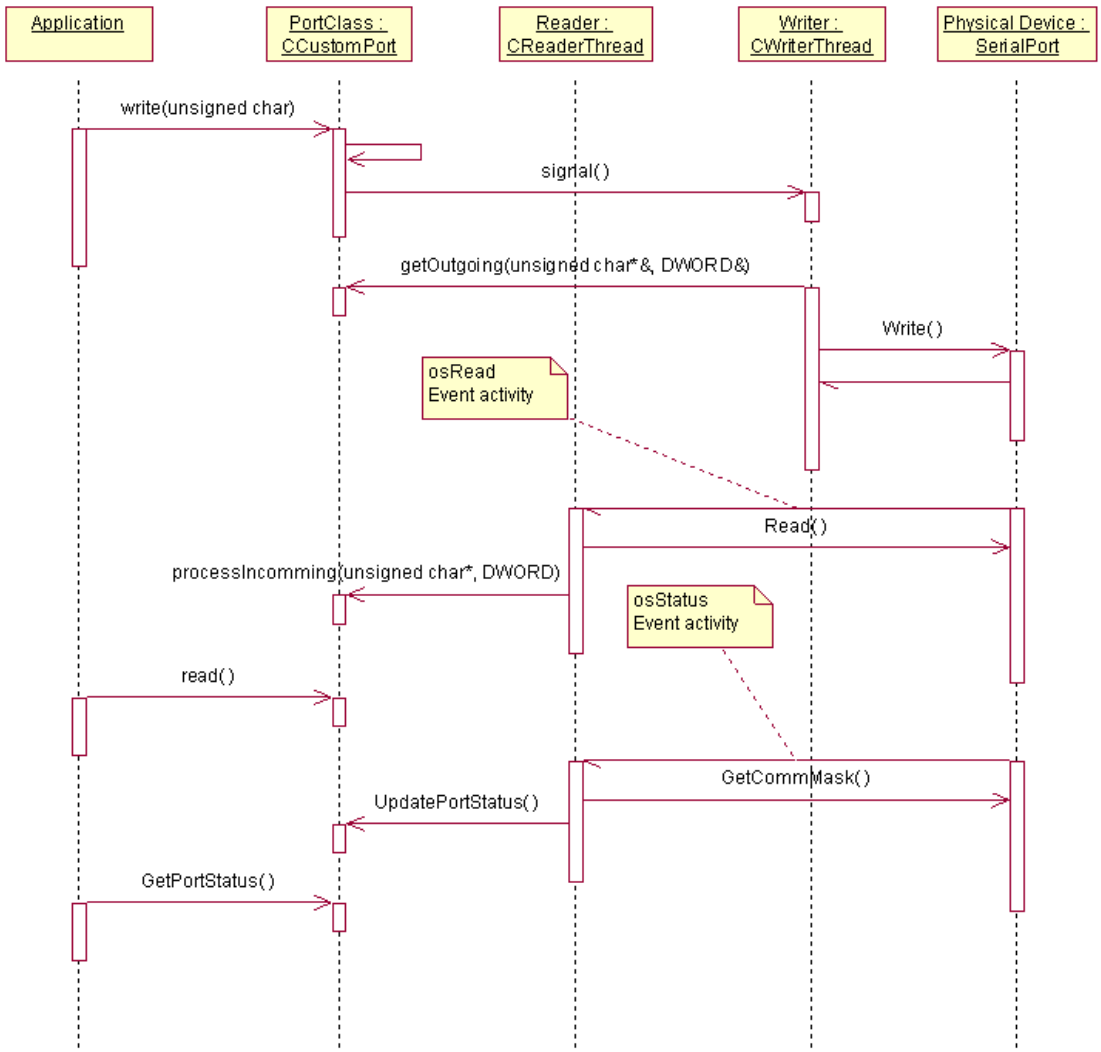


Figure 17 Serial port operation sequence diagram

The class diagram of the component (including the described below descendant classes) is presented below.

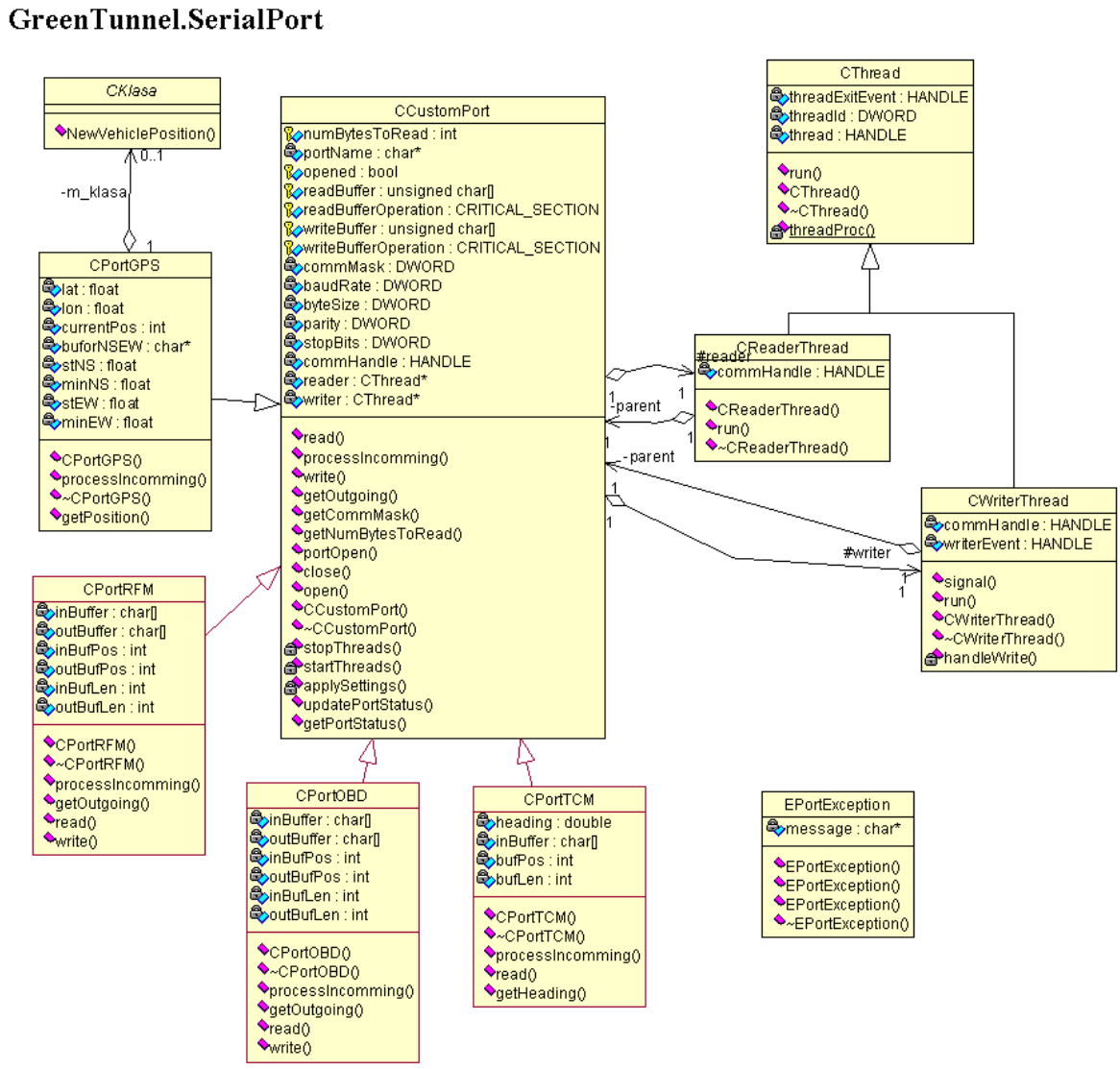


Figure 18 Serial Port class diagram

7.7.2 The CPortGPS subclass

The CPortCPS class is provided for reading data from the GPS receiver. In this application only the latitude and longitude reported by the receiver is required, so the class decodes only the GPRMC sentences, as only those sentences carry interesting data (vehicle position) (about GPS

data format see section 7.4.2). The algorithm of extracting the data is simple. It is composed of a few steps:

1. Find the beginning of the frame (the "$" sign)
2. Check if the talker ID equals to "GP"
3. Check if the sentence ID equals to "RMC"
4. Extract the position from fields 3, 4, 5 and 6
5. Verify the checksum

If any of the steps fails, it is necessary to wait for the next frame to arrive and to restart the algorithm. The algorithm is implemented in the processIncomming() method of the descendant class CPortGPS. Current position can be obtained by calling the getPosition() method. Additionally, an abstract class CKlasa has been created to provide an interface for other classes, which would like to listen to position updates. Such class can register in the CPortGPS, and once new position is received, the NewVehiclePosition() method is called.

7.7.3 The CPortTCM subclass

The PortTCM subclass of the CCustomPort class in deigned to receive and decode the data from the magneto-inductive compass described earlier. As the output word format may vary, the parser should search for the "C" character and extract the next field, as this gives the information about current compass direction.

```
$C328.3T22.3*checksum<cr><lf>
```

The algorithm of extracting data is as follows:

1. find the beginning of the frame (the "$" character)
2. extract the following numbers and the "." character and convert them to number
3. calculate and verify checksum

If the algorithm fails on any of the steps, the frame is rejected. The algorithm is implemented inside the processIncomming() method. Current compass heading can be obtained by calling the getHeading() method.

7.7.4 The CPortOBD subclass

This class is a simple extension of the base class, allowing ASCII string to be sent over the RS232 link. The implementation overrides the read(), write(), processIncomming() and getOutgoing() methods to provide buffers for character string and appropriate handling for both printing and non-printing ASCII characters.

7.7.5 The CPortRFM subclass

This class is much like the above class in the way in handles ASCII characters. However, the RFM transceivers require some additional AT commands for initialization and sending. The initialization string is sent after successful port opening, but before the actual transmission starts.

7.8 Messages exchanged between modules

7.8.1 Communication between the DAM and the ODM

CDeleteRequest

A request deletion message.

Field name	Field type	Description
m_stationID	int	ID of the station which deletes a request.
m_requestID	int	ID of the removed request.

CLogin

User login request.

Field name	Field type	Description
m_login	CString	Login entered by the user
m_password	CString	Password entered by the user.

CLoginAck

User login acknowledge.

Field name	Field type	Description
m_login_OK	int	The login process succeeded
m_user_id_user	int	User's identifier
m_user_id_role	int	User's Role ID
m_user_id_boss	int	User's boss ID
m_user_name	CString	User's name
m_user_surname	CString	User's surname
m_user_phone	CString	User's phone number.
m_station_id_station	int	User's Station ID

m_station_name	CString	Station's name
m_station_ambulance	BOOL	Station's properties
m_station_police	BOOL	
m_station_fire	BOOL	
m_station_id_arc	int	Localization of the station
m_station_shift	int	

CLogout

This message has no parameters; its purpose is only to signal an event.

CObstacleMsg

New obstacle definition.

Field name	Field type	Description
m_action	{add, edit, del}	The desired action
m_start	CString	The beginning timestamp
m_endUnknown	BOOL	If the end time is known
m_end	CString	The ending timestamp
m_description	CString	Obstacle description
m_nItems	int	Number of items in m_list
m_list []	struct { int idArc; int idObstacle;}	The arc or the obstacle identifier.

CObstacleAck

Obstacle definition acknowledge.

Field name	Field type	Description
m_OK	BOOL	ObstacleMsg processed successfully.

CPGTMsg

New Permanent Green Tunnel Definition.

Field name	Field type	Description
m_userID	int	ID of the user, who added this PGT
m_stationID	int	ID of the user's station
m_description	CString	PGT description
m_nPoints	int	Number of PGT points
m_points []	struct { int arcID; int shift; BOOL hasTimesamp; CString timestamp	Requested localizations of the PGT points with optional timestamps.

CPGTAck

Permanent Green Tunnel definition acknowledge.

Field name	Field type	Description
m_nPoints	int	Number of PGT points
m_points []	struct { int arcID; int shift; BOOL hasTimesamp; CString timestamp	Accepted PGT points

CRequestMsg

New request definition.

Field name	Field type	Description
m_userID	int	ID of the user, who added this PGT
m_stationID	int	ID of the user's station
m_arcID	int	Request target localization
m_shift	int	
m_regions	BOOL	Select vehicles from multiple regions
m_nVehicles	int	Number of entries in m_vehicles
m_vehicles []	struct { int vehicleID; int count; }	The id of the requested vehicle type and the number of vehicles requested

CRequestAck

New request definition acknowledge.

Field name	Field type	Description
m_AllOK	BOOL	All vehicles were allocated
m_nVehicles	int	Number of entries in m_vehicles
m_vehicles []	struct { int vehicleID; int count; }	If m_AllOK is FALSE, this contains the numbers of vehicles, that couldn't be allocated

CRetTunnelMsg

New Return Green Tunnel definition.

Field name	Field type	Description
m_userID	int	ID of the user, who added this PGT
m_stationID	int	ID of the user's station
m_arcID	int	The localization of the return tunnel's target
m_shift	int	
m_vehicleID	int	The ID of the vehicle following the tunnel

CTrafficUpdate

Update Message concerning new traffic congestion.

Field name	Field type	Description
m_nItems	int	Number of entries in m_items
m_items []	struct { int arcID; int cars; }	The traffic intensity of given arc

CTunnelError

A message informing that some Green Tunnels could not be set.

Field name	Field type	Description
m_nItems	int	Number of entries in m_items
m_items []	int[]	Identifiers of the tunnels that could not be set.

CUserMsg

User account add, delete or edit message.

Field name	Field type	Description
m_userID	int	ID of the user, who added this PGT
m_stationID	int	ID of the user's station
m_bossID	int	ID of the user's boss
m_roleID	int	ID of the user's role
m_name	CString	User's name
m_surname	CString	User's surname
m_phone	CString	User's phone number
m_login	CString	User's login
m_password	CString	User's password
m_action	{add, edit, del}	The action description

CUserAck

User Message acknowledge.

Field name	Field type	Description
m_OK	BOOL	UserMsg processed successfully

CObstacleUpdate

Update Message concerning messages that should be added, changed or removed.

Field name	Field type	Description
m_action	{add, edit, del}	The action description
m_nItems	int	Number of entries in m_items
m_items []	struct {	Description, localization, start and end times of each obstacle

	CString desc; CString end; int id_arc; int id_obstacle; CString start; }	

CVehicleMsg

Vehicle add, delete or edit message.

Field name	Field type	Description
m_userID	int	ID of the user, who added this PGT
m_stationID	int	ID of the user's station
m_vehicleID	int	ID of the vehicle
m_vehicleTypeID	int	ID of the vehicle's type
m_arcID	int	Current localization of the vehicle
m_shift	int	
m_registration	CString	Vehicle's registration number
m_action	{add, edit, delete, lca, avail, unavail, ret}	The action description

CVehicleAck

Vehicle Message acknowledge.

Field name	Field type	Description
m_OK	BOOL	VehicleMsg processed successfully

CVehiclePosUpdate

Update Message concerning current vehicle position.

Field name	Field type	Description
m_count	int	Number of entries in m_objects
m_objects []	struct { int vehicleID; int arcID; int shift; }	New position o the vehicle of a given type

CVehicleRoute

Information about green tunnels set in the city.

Field name	Field type	Description
m_nArcs	int	Number of entries in m_arcs
m_arcs []	int[]	Identifiers of the arcs that are parts of some green tunnels.

CVehicleStateUpdate

Update Message concerning current vehicle status.

Field name	Field type	Description
m_vehicleID	int	ID of the vehicle
m_state	{“A”, “N”, “R”, “L”, “C”}	Current vehicle state

CVehicleTypeUpdate

Update Message concerning the addition of a new vehicle type.

Field name	Field type	Description
m_id_vehicle_type	int	ID of the vehicle's type
m_name	CString	Type name
m_symbol	CString	Type symbol
m_description	CString	Type description

7.8.2 Communication between the DAM and the GTM

CCalculateMsg

This message has no body; its purpose it just to signal an event and trigger green tunnels recalculation.

CLoginGTM

The Green Tunnel Module login request.

Field name	Field type	Description
m_login	CString	Login string
m_password	CString	Password string

CLogoutGTM

This message has no parameters; its purpose is only to signal an event.

CTunnelMsg

This message has no body; its purpose it just to signal an event that the recalculation of green tunnels has finished.

CTunnelError

A message informing which requests could not be processed.

Field name	Field type	Description
m_nItems	int	Number of entries in m_items
m_items []	int[]	Identifiers of the requests that could not be processed.

7.8.3 Communication between the DAM and the TS

CLoginTS

The Traffic Simulator login request.

Field name	Field type	Description
m_login	CString	Login string
m_password	CString	Password string

CLogoutTS

This message has no parameters; its purpose is only to signal an event.

CTrafficUpdate

The message has the same format as the message exchanged between the DAM and ODM.

7.8.4 Communication between the DAM and the VCM

CLoginVCM

The Vehicle Communication Module login request.

Field name	Field type	Description
m_login	CString	Login string
m_password	CString	Password string

CLoginVCMAck

The LoginVCM Message acknowledge.

Field name	Field type	Description
m_OK	BOOL	Login successfull

CLogoutVCM

This message has no parameters; its purpose is only to signal an event.

7.8.5 Communication between the CRM and the DAM

CLoginCRM

The Crossroads Module login request.

Field name	Field type	Description
m_login	CString	Login string
m_password	CString	Password string

CLoginCRMAck

The LoginCRM Message acknowledge.

Field name	Field type	Description
m_OK	BOOL	Login successfull

CLogoutCRM

This message has no parameters; its purpose is only to signal an event.

CTunnelMsg

This message has no body; its purpose it just to signal an event that the recalculation of green tunnels has finished.

7.8.6 Communication between the VM and the VCM

CLoginVM

The Vehicle Module login request.

Field name	Field type	Description
m_login	CString	Login string
m_password	CString	Password string

CLoginVMAck

The Vehicle Module login acknowledge.

Field name	Field type	Description
m_OK	BOOL	Login successful
m_vehicleID	int	ID of the vehicle
m_stationID	int	ID of the originating station
m_vehicleTypeID	int	ID of the vehicle's type
m_name	CString	Type name
m_symbol	CString	Type symbol
m_registration	CString	Vehicle's registration number

CLogoutVM

This message has no parameters; its purpose is only to signal an event.

CObstacleVM

The message informing about an obstacle detected by a vehicle driver.

Field name	Field type	Description
m_vehicleID	int	ID of the vehicle
m_stationID	int	ID of the originating station

CRetTunnelVM

The message informing that the vehicle requests the Return Green Tunnel.

Field name	Field type	Description
m_vehicleID	int	ID of the vehicle
m_stationID	int	ID of the originating station

CTunnelErrorVM

The message informing that the return tunnel request could not be processed.

Field name	Field type	Description
m_vehicleID	int	ID of the vehicle

CVehicleRoute

The new vehicle's route specification.

Field name	Field type	Description
m_vehicleID	int	ID of the vehicle
m_arcID	int	Precise target localization
m_shift	int	
m_nArcs	int	Number of entries in m_arcs
m_arcs []	int	Identifiers of the arcs on the vehicle's route

7.9 Source code statistics

Source code size:

Module name	Lines of code
Crossroads Module	1000
Data Access Module	1200
Database Access Classes	3000
Encrypted TCP Client	1000
Green Tunnel Module	1380
Map Component	3600
Messages	2000
Officer-on-Duty Module	3600
Serial port routines	600
Traffic Simulator	250
Vehicle Communication Module	300
Vehicle Module	800
Total:	15 730

The table below shows the system complexity according to McCabe's complexity measure [15]:

Condition statement	Occurrences
if	1448
for	526
case	176
while	226
and	139
or	61
Total:	2 576

7.10 Glossary

Administration & Monitoring Center (A&MC) a crucial part of the GREEN TUNNEL SYSTEM; a location where all administration and monitoring actions take place; Monitoring Module(s) and Administrtion Module(s) are installed there.

Administration Module (AM) one of the GREEN TUNNEL SYSTEM's base modules. Using this module one can easily prepare the system to start working, i.e. create new user accounts, generate RSA private and public keys, update city plan, set security level of the whole system, etc.

Asymmetric algorithm in cryptography, an asymmetric algorithm uses a pair of different cryptographic keys to encrypt and decrypt the plain text. Typically, the two keys are related mathematically; a message encrypted by the algorithm using one key can be decrypted by the same algorithm using the other. In a sense, one key "locks" a lock (encrypts); but a different key is required to unlock it (decrypt). Example: RSA algorithm.

Bellman-Ford algorithm an efficient algorithm to find the shortest paths from a single source vertex to all other vertices in a weighted, directed graph. Weights may be negative.

Biscuit PC PC (Personal Computer) which size is reduced to fit small card used in embedded systems; besides standard I/O ports, it can be equipped with special feathures as well.

Crossroads Module (CRM) one of the GREEN TUNNEL SYSTEM's base modules. The aim of this module is to control selected traffic light controllers in order to create desired Green Tunnel. It also traces position of vehicles assigned to Green Tunnels to ensure proper moment of setting up a green light on a

crossroad, as well as restoring "normal" traffic light controlling program after the vehicle pass the junction.

Cryptographic keys a cryptographic key is a relatively small amount of information that is used by a cryptographic algorithm to "customize" the transformation of plaintext into cyphertext (during encryption) or from cyphertext to plaintext (during decryption). The same algorithm and plaintext, but with a different key will produce a quite different cyphertext, and so for decryption as well. If the decryption key is lost, encrypted data will not in practice be convertible back to its original form – at least for high quality encryption algorithms and large enough key sizes. Thus, the security of a cryptographic key in most cases relies on its being kept secret: hence the alternative name secret key.

Data Access Module (DAM) one of the GREEN TUNNEL SYSTEM's base modules. This module ensures connection between Green Tunnel Database and other modules. It uses encryption methods to transmit data and carry out authorization process.

Data consistency it is a state when all the data in database has one specified value for all the users.

Dijkstra's algorithm An algorithm to find the shortest paths from a single source vertex to all other vertices in a weighted, directed graph. All weights must be nonnegative.

Directed graph a graph that has vertices (or nodes) connected by directed arcs. It is important that if node n_1 has connection with node n_2 does not imply a connection between n_2 and n_1.

Direction priority a priority set on a crossroad's direction. If one direction has higher priority than the rest ones, it means that green light will be turned on more frequently and will last longer on it.

Distributed Internet application application that is run on many computers connected with each other via Internet. It characterizes with many copies of the database information and independent action occurrence.

Emergency Department (ED) a crucial part of the GREEN TUNNEL SYSTEM; a location where Officer-on-Duty Modules are installed. Usually ED are Health Care Centers, City Fire Departments, Police Departments, or other emergency services that accidents are reported to.

Emergency services any kind of service, which takes steps when an accident occurs, or where accidents can be reported to.

Entity-relationship diagram an entity-relationship diagram is a way of representing the structure of a relational database. An entity represents a discrete object. Entities can be thought of (roughly) as nouns. Examples: a computer, an employee, a song, a mathematical theorem. A relationship captures how two or more entities are related to one another. Relationships can be thought of (again, roughly) as verbs. Examples: an owns relation between a company and a computer, a supervises relation between an employee and a department, a performs relation between an artist and a song, a proved relation between a mathematician and a theorem.

eXtreme Programming (XP) a method in or approach to software engineering, formulated by Kent Beck. In general, eXtreme Programming is believed to be useful for small teams under 10 persons.

FIFO order First In, First Out is the principle of a queue: what comes in first is handled first, what comes in next waits until the first is finished, etc.

Floyd-Warshall algorithm an algorithm to solve the all pairs shortest path problem in a weighted, directed graph by multiplying an adjacency-matrix representation of the graph multiple times. The edges may have negative weights, but no negative weight cycles.

Fuzzy position an area when an object (e.g. a vehicle) can be found with a given degree of likeliness.

GPS The Global Positioning System, usually called GPS, and originally named NAVSTAR, is an intermediate circular orbit (ICO) satellite navigation system used for determining one's precise location and providing a highly accurate time reference almost anywhere on Earth.

GPS emulator a program that outputs data which could be obtained from a real GPS device.

Green Tunnel „a wave" of green traffic lights at the crossroads located on the possible shortest route of a vehicle to a place of an accident. It helps the vehicle to get to the place of an accident in the possible shortest time, as it unloads heavy traffic on the vehicle's route and gives it the movement priority on the crossroads.

Green Tunnel Control Center a crucial part of the GREEN TUNNEL SYSTEM; a location where all calculations concerning traffic intensity, arrangement of the Green Tunnels and vehicles' routes take place. It is also a place where Green Tunnel Database is kept.

Green Tunnel Database (GTDB) a database that stores all crucial information about vehicles, Green Tunnels, traffic intensity, city plan, etc., which is used by the GREEN TUNNEL SYSTEM.

Green Tunnel Module (GTM) one of the GREEN TUNNEL SYSTEM's base modules. This module is responsible for calculating the arrangements of all the Green Tunnels in the city, as well as routes for the vehicles. It takes into consideration such data as: city map, traffic intensity, vehicles' fuzzy positions and its states, time of arrival to the place of an accident.

Green Tunnel System (GTS) a name of a system, that supports everyday work of all emergency services in the city, helping them to get to the place of an accident as fast as possible. It makes use of existing traffic infrastructure such as traffic lights controllers, traffic detectors, etc.

Group model one of many models used in modeling traffic and behavior of drivers and pedestrians.

Hash table A dictionary in which keys are mapped to array positions by a hash function. Having more than one key map to the same position is called a collision. There are many ways to resolve collisions, but they may be divided into open addressing, in which all elements are kept within the table, and chaining, in which additional data structures are used.

Iteration in eXtreme programming a little but fully functional increment of the developing program/software.

Johnson's algorithm An algorithm to solve the all pairs shortest path problem in a sparse weighted, directed graph.

Key exchange process automatic process of establishing one key used in symmetric algorithm in such a way, that even if all transmitted data are know to third party, it can not reconstruct correct key.

Local adaptation light controlling scheme a scheme in order which traffic light controller to, concerning data of local (limited to one junction) traffic intensity and direction priorities.

Message-passing model a model that describes a way of exchanging data between separate modules of the system, in which all communication between them happens only with the use of sending and receiving messages. Other model of module communication is shared memory model.

Model of communication a model according to which data are transmitted between stationary and mobile part of the system (see communication between Vehicle Communication Station and Vehicle Modules).

Monitor a program that checks if monitored program works correctly. When some errors occur in monitored application, monitor can react and repair these faults or pause malfunctioning application and inform a user about it.

Monitoring Module (MM) one of the GREEN TUNNEL SYSTEM's base modules. With the use of this module user can remotely check state of a vehicle or a traffic light. It is also possible to see traffic intensity on the streets in a city, and set up Green Tunnels, too.

Multithreaded application which operates in a lot of threads.

Mutual exclusion used in concurrent programming to avoid the concurrent use of unshareable resources by pieces of computer code called critical sections.

OBD a standard for vehicle diagnostics systems, introduced in the mid-'90s, provides almost complete engine control and also monitors parts of the chassis, body and accessory devices, as well as the diagnostic control network of the car.

Obstacle in the GREEN TUNNEL SYSTEM any of obstacle that makes road traffic almost or totally impossible, such as traffic jam, road-works, natural accidents (flood, fallen tree), etc.

ODBC (Open Database Connectivity) a standard software API for connecting to database management systems of various brands.

Officer-on-Duty Module (ODM) one of the GREEN TUNNEL SYSTEM's base modules. This module is installed in Emergency Departments, where officers-on-duty register accidents, makes requests for a set of vehicles or for a Green Tunnel, etc. It also show a city map with traffic congestion, assigned and available vehicles, or Green Tunnels plotted on it.

Permanent Green Tunnel (PGT) a type of Green Tunnel that is known to appear relatively long before it is really set up. It may be useful for special-purpose transportation, where it is known in advance that a vehicle will follow a certain route at a certain time.

Private key a key used in public key cryptography, which is used to decrypt cyphertext and must be kept secret ("private").

Public key a key used in public key cryptography, which is used to encrypt plaintext and can me widely known

Public key cryptography asymmetric-key cryptography, also known as public-key cryptography, is a form of cryptography in which asymmetric key algorithms are used for encryption, dignature, etc. In these algorithms, one key is used to encrypt a message and another is used to decrypt it, or one key is used to sign a message and another is used to verify the signature. The key used to decrypt or sign must be kept secret ("private") and cannot (so algorithm designers hope) be derived from the public key, which is used to encrypt or verify, and which may be known to any.

Request in the GREEN TUNNEL SYSTEM a demand submitted by officer-on-duty (using Officer-on-Duty Module) to the Green Tunnel Control Center for an assignment of the vehicles to an accident and creation of Green Tunnel for them.

Return Green Tunnel (RGT) a type of Green Tunnel. RGT is set only for a demand of vehicle driver (but the officer-on-duty requests for it) when he needs to have a quick way back, i.e. to transport dangerous criminal to Police Department or casualties to the hospital, as fast as possible.

Rijndael algorithm Rijndael is a block cipher, designed by Joan Daemen and Vincent Rijmen as a candidate algorithm for the AES. The cipher has a variable block length and key length. The design of Rijndael was strongly influenced by the design of the block cipher Square.

RSA algorithm RSA is an asymmetric algorithm for public key cryptography, widely used in electronic commerce. The algorithm was described in 1977 by Ron Rivest, Adi Shamir and Len Adleman; the letters RSA are the initials of their surnames.

Shared memory model a model that describes a way of exchanging data between separate modules of the system, in which all communication between them happens only with the use of shared memory. Other model of module communication is message-passing model.

Shortest path problem the problem of finding the shortest path from one vertex in a graph to another vertex. "Shortest" may be least number of edges, least total weight, etc. Example: Bellman-Ford algorithm, Dijkstra's algorithm, Johnson's algorithm, Floyd-Warshall algorithm.

Single-producer-multi-consumer thread-safe buffer an intermediate storage place for data, which are written by one thread and read by one or more threads;

necessary synchronization objects and methods for operation in multiple threads are provided.

Six hats technique a kind of role playing game, helpful in the process of software designing. Used when continuous contact with clients (which specifies their requirements) is impossible, but a requirement problem has to be solved.

Spike solution a small, informal software experiment which leads to the solution of a bigger problem.

Symmetric algorithm a symmetric-key algorithm is an algorithm for cryptography that uses the same cryptographic key to encrypt and decrypt the message. Example: Rijndael algorithm.

System administrator describes the role of a person whose job includes maintenance duties for the long-term support of a computer system.

Table of priorities a table which stores directions' priorities stored in a memory of each traffic lights controller.

Thread-safe operation operation in multiple threads without any problems such as access violation, guaranteed by synchronization methods and objects (e.g. critical sections, semaphores etc.)

Timestamp a date and time associated with an object; uniquely identifies certain moment of time.

Traffic Control Center (TCC) a place in a city where all information concerning traffic congestion, state of a traffic lights on every junction in a city, etc. comes to. It is not required to exists for the GREEN TUNNEL SYSTEM.

Traffic light controller a technical device which controls traffic lights on a crossroad according to a specified program. Several types of control programs can be

distinguished, such as: constant time green light, local adaptation, local adaptation with direction priorities, etc.

Traffic Module (TM) one of the GREEN TUNNEL SYSTEM's base modules. This module is responsible for gathering data concerning traffic intensity on a junctions directly from crossroads or indirectly from Traffic Control Center, and, after the process of gathering, placing it in the Green Tunnel Database.

Traffic Simulator (TS) a module that was very helpful during development of the GREEN TUNNEL SYSTEM. Its task was to generate traffic congestion for a city stored in Green Tunnel Database, as we had only limited access to data from a real city.

Type of vehicle type of vehicle is one of requests' parameters – officers on duty decide only which type of vehicle (fire car or a specialized ambulance) have to be send, and the GREEN TUNNEL SYSTEM determines a vehicle which have to be send to an accident.

User account a set of information (e.g. name, surname, address, privileges, roles, permissions, passwords) concerning the user in a system; setting up an account gives the user a possibility to use the system

Vehicle Communication Module (VCM) one of the GREEN TUNNEL SYSTEM's base modules. This module provides communication with vehicles (i.e. Vehicle Modules) using proper communication model. It is installed in Vehicle Communication Station.

Vehicle Communication Station (VCS) a place where communication with vehicles is held. With the use of Vehicle Communication Module data transmission between Vehicle Modules and Green Tunnel Database is ensured. Number of Vehicle Communication Stations in a city is limited only by used communication model.

Vehicle Module (VM) one of the GREEN TUNNEL SYSTEM's base modules. The only one part of the system which is mobile, as it is installed in each vehicle. Its task is to interact with a driver (display him route, gather information about vehicle state, etc.) and communicate with Vehicle Communication Station. Another important issue is that when GPS signal is lost, VM is responsible for localizing vehicle with the use of onboard localization system described in section 3.2.5.1.

Vehicle state information about availability of each vehicle. Possible states are: "A" – available (when it is possible to assign a vehicle to a request), "U" – unavailable (when vehicle is already assigned to a request or it is e.g. out of order), "L" – less critical action (when vehicle is assigned to less critical action and can be assigned to a more critical one), "R" – Return Green Tunnel (when vehicle is assigned to a Return Green Tunnel and cannot be assigned to any action).

Waterfall software developing model The Waterfall model is a software development model first proposed in 1970 by W. W. Royce, in which development proceeds linearly through the phases of requirements analysis, design, implementation, testing (validation), integration and maintenance.

Wire-tapping describes the monitoring of the connection by a third party, often by covert means. It is a way of stealing the data.