VEWE: A Vehicle ECU Wireless Emulation Tool Supporting OBD-II Communication and Geopositioning

Óscar Alvear, Carlos T. Calafate, Juan-Carlos Cano, and Pietro Manzoni

Department of Computer Engineering (DISCA),
Universitat Politècnica de València, Spain
oalvear@gmail.com, {calafate,jucano,pmanzoni}@disca.upv.es

Abstract. Almost all the vehicles built during the last decade integrate an On Board Diagnostic (OBD-II) interface, through which it is possible to monitor and manage multiple operational parameters. In the past few years, Bluetooth OBD-II devices have been introduced in the market to facilitate connection to mobile devices. With the increased use of these devices, many applications for real-time control and monitoring of different parameters are being developed in the automotive sector. This infrastructure has opened a broad research area related to "smart driving". The main problem in the development and testing of these applications is the need to debug and validate them using different vehicles, under different configurations and scenarios. Our proposal to address this problem is VEWE: Vehicle Wireless Emulator offering realistic vehicle dynamics, which allows testing the correctness and the performance of mobile applications using off-the-shelf computers, thereby speeding up the development time and lowering costs. Another useful functionality of VEWE is the emulation of Geo-positions in Android systems through a GPS-Emulator. By combining both functionalities, VEWE provides a complete and flexible development environment for mobile vehicular applications.

1 Introduction

Mobile devices have experienced a technological breakthrough in recent years, evolving towards high performance terminals with multi-core microprocessors, being smartphones a clear representative exponent of this trend. In addition, the On Board Diagnostics (OBD-II) [1] standard, available since 1994, has recently become an enabling technology for in-vehicle applications due to the appearance of Bluetooth OBD-II connectors [2]. These connectors enable a transparent connectivity between the mobile device and the engine's Electronic Control Unit (ECU).

The range of possibilities that arise when combining cars and smartphones is endless, allowing, for example, diagnosing the car via mobile devices which assume the tasks that are typically performed by the On Board Unit (OBU) of the vehicle, sending the collected data to a platform where diagnosis and vehicle

S. Guo et al. (Eds.): ADHOC-NOW 2014, LNCS 8487, pp. 432–445, 2014.

maintenance can be done, detecting possible failures automatically, or developing solutions to automatically analyze driver behavior and suggest corrective actions when necessary.

Currently, it is already possible to find several smartphone-based applications that rely on OBD-II communications [3]. However, the development of these applications is costly since the developer has to deal with real ECUs from different manufacturers in order to test and debug the applications, usually requiring taking the vehicle for short test trips. To avoid this requirement, and to speed-up the development process, in this paper we propose VEWE, a vehicle wireless emulator that includes map-based mobility modeling, and a simulated engine ECU accessible through a wireless interface. The functionality provided by VEWE allows the developers to test OBD-II based smartphone applications as if moving in a real vehicle.

This paper is organized as follows: in section 2 we present some related works, evidencing how our work differs from previous ones. In section 3 we provide an overview of the OBD-II standard. The VEWE solution is introduced in section 4, and technical details are provided in section 5. Finally, section 6 presents the main conclusions of this paper.

2 Related Works

Currently we can find a broad range of smartphone applications able to communicate with a vehicle's ECU to provide enhanced services to drivers in the scope of Intelligent Transportation Systems (ITS) area.

Torque [4] is a well known Android application that allows monitoring all sorts of parameters available through the OBD-II interface (e.g. vehicle speed, engine RPM, Fuel pressure), offering the user a comfortable visualization by allowing to personalize the actual size and position of data displays.

Teng et al. [5] implement an Android-based mobile device platform able to read data on the vehicle ECUs; results are graphically displayed as a virtual instrument on the mobile panel.

Meseguer et al. proposed DrivingStyles [6], a solution combining an Android-based application and a web platform that is able to determine the type of road where the driver is circulating, as well as his driving habits. Its main goal is to help promoting a safer and more ecological driving style by making drivers more conscious about their behavior on the road.

Zaldivar et al. [7] proposed an Android-based application that monitors the vehicle through the On Board Diagnostics (OBD-II) interface, being able to detect accidents and sending details about the accident to pre-defined destinations through either e-mail or SMS; these tasks are immediately followed by an automatic phone call to the emergency services.

Tahat et al. [8] developed an Android application able to monitor the vehicle's fuel consumption and other vital electromechanical parameters. Data can also be sent to the vehicle's manufacturer maintenance department, allowing to detect and predict vehicle faults while moving.

The aforementioned research works highlight the interest in developing smartphone applications that interact with vehicles. Nevertheless, since available OBD-II emulation tools are too simple and lack GPS integration [9,10], developers should perform test driving to evaluate their proposals, which is expensive and time consuming. The platform presented in this paper aims to simplify and accelerate development by modeling vehicle dynamics in detail and allowing to jointly emulate OBD-II communications and geopositioning, thus meeting the basic requirements of mobile application developers working in this field.

3 The OBD-II Standard

The On-board Diagnostic (OBD) standards [11] were developed in the USA to detect car engine problems that can provoke an increase of CO2 gas emission levels beyond acceptable limits. To achieve this purpose, the system is constantly monitoring the different elements related to gas emissions, including engine management functions, being a powerful tool to diagnose problems on vehicles' electrical systems. When a failure is detected, the system must store it in its memory so that technicians may analyze it later on.

The first OBD standard, known as OBD-I, defined just a few parameters to monitor, and did not establish a specific emission level for vehicles. Thus, failures resulted in just a visual warning to the driver and the storage of the error. The second generation of OBD, known as OBD-II, standardizes different elements such as the connector used for diagnostic, the electrical signaling protocols, and the message format. Additionally, it defines a list of parameters that can be monitored, assigning a specific code to each parameter. A detailed list of DTCs (Diagnostic Trouble Codes) is also defined in the standard (see [11]).

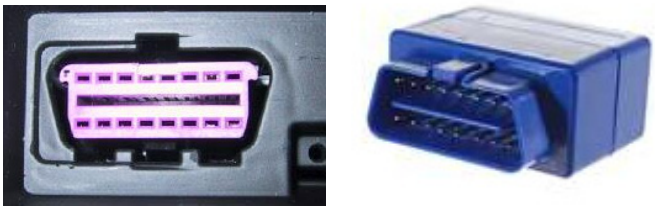
Several operating modes are defined by the OBD-II standard to provide an easier interaction with the system, and achieve the desired functionality. Most automobile manufacturers have introduced additional operation modes that are specific to their own vehicles, thus offering a full control of the available functionality.

The European version of the OBD-II standard, known as EOBD, is mandatory for all gasoline and diesel vehicles since 2001 and 2003, respectively. Despite it introduces some small improvements, EOBD strongly resembles OBD-II, sharing the same connectors and interfaces.

Figure 1 shows an example of both male and female OBD-II connectors. In particular, the male connector shown in the figure is part of a Bluetooth-enabled OBD-II device that offers a bridge between the vehicle's internal bus and a smartphone using a Bluetooth connection.

3.1 Communication Protocols

Although the physical interface is well defined, the communications protocol varies depending on the manufacturer. Different protocols are available: (i) SAE J1850 PWM (Pulse-Width Modulation) [12], (ii) SAE J1850 VPW (Variable



(a) Female connector (b) Male connector.

Fig. 1. Example of a) an in-vehicle OBD-II female connector, and b) a Bluetooth-enabled OBD-II device with male connector

Pulse Width) [12], (iii) ISO 9141-2 [13], (iv) ISO 14230 KWP2000 (Keyword Protocol 2000) [1], and (v) ISO 15765 CAN [11]; these protocols present significant differences between them in terms of the electrical pin assignments. Notice that most vehicles implement only one of these protocols. For instance, Chrysler uses the ISO 9141-2 protocol, General Motors uses SAE J1850 VPW, and Ford uses SAE J1850 PWM.

3.2 Diagnostic Trouble Codes (DTCs)

Diagnostic Trouble Codes were standardized in document ISO 15031-6 [14], and allows engine technicians to easily determine why a vehicle is malfunctioning using generic scanners. The proposed format assigns alphanumeric codes to the different causes of failure, although extensions to the standard are allowed to support manufacturer-specific failures.

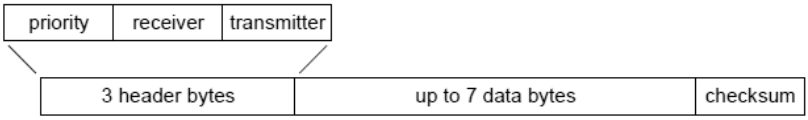
3.3 OBD Message Formats

The OBD system was designed to offer a flexible communications system. Message delivery among different devices requires defining the type of message to be delivered, along with the transmitter and the receiver devices. The adoption of different message priorities is also supported in order to make sure that critical information is processed first.

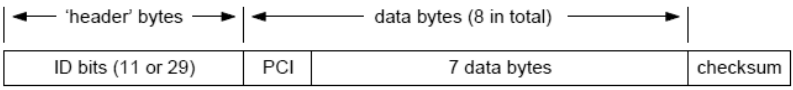
However, depending on the protocol using by each vehicle, the format of this message may vary slightly (see figure 2). Notice that both frame formats allow up to 7 data bytes, and they also include a checksum field in order to detect any transmission errors.

3.4 OBD-II PIDs

The OBD-II PIDs (OnBoard Diagnostics Parameter IDs) are identification codes of the different parameters that can be measured in a vehicle. The OBD-II standard, defined by the Society of Automotive Engineers as SAE J1979, defines



(a) Frame format adopted by the SAE J1850, ISO 9141-2 and ISO 14230-4 standards.



(b) Frame format adopted by the ISO 15765-4 (CAN) standard.

Fig. 2. OBD frame formats

Table 1. OBD-II operation modes

Mode	Description
01	Show current data
02	Show freeze frame data
03	Show stored Diagnostic Trouble Codes
04	Clear Diagnostic Trouble Codes and stored values
05	Test results, oxygen sensor monitoring (non CAN only)
06	Test results, other component/system monitoring (Test results, oxygen sensor monitoring for CAN only)
07	Show pending Diagnostic Trouble Codes (detected during current or last driving cycle)
08	Control operation of on-board component/system
09	Request vehicle information
0A	Permanent Diagnostic Trouble Codes (DTCs) (Cleared DTCs)

some parameters, although vehicle manufacturers usually introduce their own codes.

As shown in table 1, the SAE J1979 standard defined some operation modes for accessing OBD-II information.

The SAE J1979 standard also defines several OBD-II PIDs for the different modes of operation. We focus on mode 01 since it provides access to the current vehicle data. Numerical values are sent and received in hexadecimal format. Table 2 shows some codes along with their description and the parameters required to interpret them.

The "PID" column indicates the code identifier, the “Size” column indicates the length in number of bytes, the "Description" column describes the code, columns "Min" and "Max" indicate the minimum and maximum values, respectively, column "Units" indicates the units in which the code is read, and finally the "Formula" column shows which formula must be applied to interpret the received value.

Table 2. OBD-II PIDs

PID	Size	Description	Min	Max	Units	Formula
00	4	Supported PIDs				
05	1	Engine coolant temperature	-40	215	°C	A-40
0A	1	Fuel pressure	0	765	kPa (gauge)	A*3
0C	2	Engine RPM	0	16,383.75	rpm	((A*256)+B)/4
0D	1	Vehicle speed	0	255	km/h	A

When communicating via OBD-II two codes are required: one indicating the mode, and another one to specify the monitored parameter. For example, when requesting the vehicle speed, the identifier is "010D" where "01" indicates the mode, and "0D" the speed parameter. The response to this request would be "410D XX"; notice that the first digit identifier is changed from "0" to "4" to indicate an answer, and "XX" is the returned value.

In the "Formula" column, letter A represents the first byte returned, letter B represents the second byte, letter C the third byte, and so on.

4 VEWE: Vehicle ECU Wireless Emulator

VEWE is a solution developed to simulate vehicular behavior, and offer access to the vehicle's ECU parameters through a Bluetooth OBD-II interface. By using VEWE, the developers of OBD-II based smartphone applications are able to test them as if they were moving in a real vehicle, thereby speeding up the development time and lowering costs.

VEWE is able to generate all the parameters of a moving vehicle, and it allows the user to control the vehicle's mobility within a map through a joystick, as well as manually setting new parameters and have control over the Bluetooth communications. Map information is obtained from OpenStreetMap [15].

The VEWE platform is composed of three components: the main application, called VEWE server, and two Android based applications: GPSEmulator and OBDIICapture. Together, they provide a complete test system for Android-based mobile vehicular applications.

VEWE has the following features:

Multiplatform. It was developed in Java, using the Bluecove library for Bluetooth connectivity [16] and JMapViewer library for the map viewing functionality, making it a multiplatform system fully functional on both Windows and Linux operating systems.

Friendly graphical interface. It has a friendly and intuitive graphical user interface, developed using the Java Swing library. The user simply "turns on" the vehicle, and then it controls its path over a map (taken from OpenStreetMap) using a joystick.

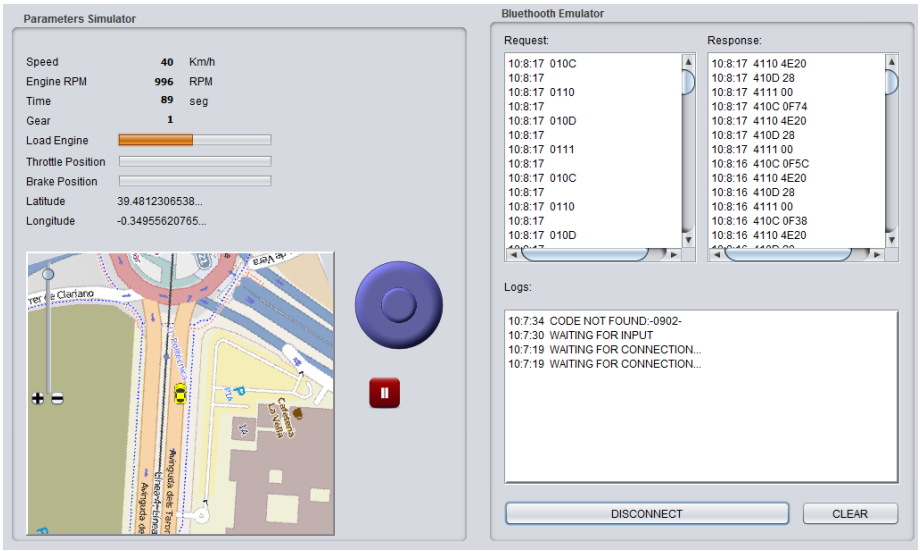


Fig. 3. VEWE Server application including the vehicle mobility simulator (left) and the OBD-II/Bluetooth connection manager (right)

Flexible. You can add or remove different simulated parameters, as well as specify the formula for calculating their value; other input parameters can be referenced in such a formula.

Geocoding simulation. Using the GPSEmulator application, we can emulate geopositions on any Android system based on data generated by VEWE in real-time. GPS coordinates are transferred between the VEWE Server and the GPSEmulator via Bluetooth.

Control of Bluetooth/OBD-II communications. The applications displays, in real-time, all the information exchanged between the simulator and the mobile device, as well as all the events generated therein. This is useful to debug OBD-II based applications being developed.

4.1 VEWE Server

The VEWE Server is the main Java application, being responsible for simulating the behavior of a moving vehicle, its engine Electronic Control Unit (ECU), and its location. For this endeavor it relies on different components: (1) Vehicle Mobility Simulator, (2) Map Manager, (3) Engine ECU, and (4) an OBD-II Emulator.

Concerning the Vehicle Mobility Simulator (1), this component is responsible for simulating a real vehicle, and dynamically determining the value of all relevant parameters, including those registered in the engine ECU. Among these parameters we can find:

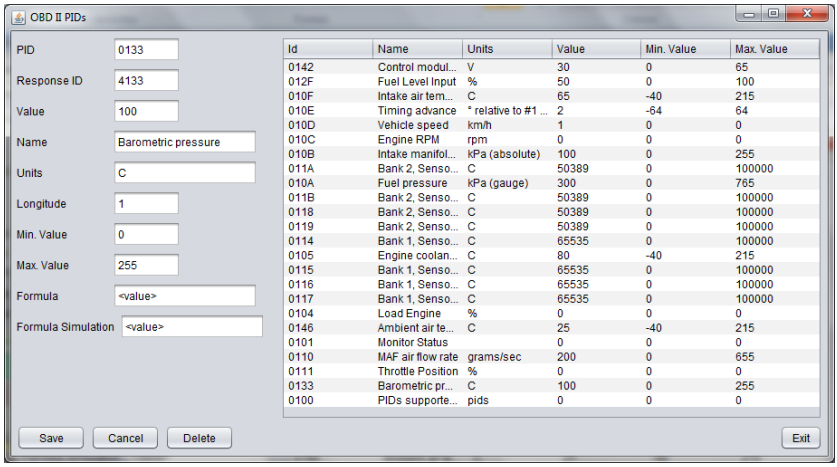


Fig. 4. VEWE OBD-II parameter management

- **Speed:** Updated according to the simulated vehicle speed.
- **Engine RPM:** Indicates the engine speed in terms of Revolutions Per Minute at any time.
- **Time:** Registers the time since the simulation started.
- **Gear:** Registers the current gear being used by the vehicle.
- **Engine Load:** Percentage value indicating the current load of the engine.
- **Throttle Position:** Percentage value indicating the relative position of the throttle pedal.
- **Brake Position:** Percentage value indicating the relative position of the brake pedal.
- **Latitude/Longitude:** Register the current vehicle coordinates.

The Map Manager (2) retrieves the actual street map from the OpenStreetMap platform [15], and provides a visual representation of the vehicle's position; such position is updated throughout time, and can be served to clients upon request.

Concerning the Engine ECU (3), it is a component used to store the value of all relevant PIDs being simulated, and the values stored can be requested through requests coming from the OBD-II interface. The VEWE Server also provides an OBD-II PID management interface (see figure 4), which contains a list of all parameters being simulated. From this interface it is possible to manually manipulate all the parameters handled by the Engine ECU.

The attributes associated to each parameter are:

OBD-II PID. Identifies the PID parameter and has a length of 4 hexadecimal digits, where the first two digits indicate the mode, and the following two PID identifier.

Response ID. Indicates the response ID for that specific PID, usually by changing the first digit from "0" to "4" .

Value. Indicates the value of the PID at that time. When a new parameter is introduced, this attribute indicates the default parameter value.
Name. Indicates the name or description of the PID.
Units. Specifies the unit associated to the PID value.
Length. States the size, in number of bytes, for each PID.
Min Value. Indicates the minimum value allowed for the PID.
Max Value. Indicates the maximum value allowed for the PID.
Formula. Indicates the formula used to transform the PID value into OBD-II format.
Simulation Formula. Indicates the formula used for calculating the PID value based on other PID values (if applicable).

Finally, regarding the Bluetooth OBD-II interface emulator (4), it is the component is responsible for controlling the Bluetooth connection. In particular, it creates and maintains the OBD-II Bluetooth connection, and registers all data sent through the established Bluetooth channel. The registered data is split into three parts:

1. Request: Shows the data requests received from clients.
2. Response: Displays the system responses returned to clients.
3. Logs: Registers important events related to the connection, such as connection status, missing codes, etc.

Besides the VEWE server application described above, our solution also includes two Android components - GPSEmulator and OBDIICapture -, which have quite different goals. These are described below.

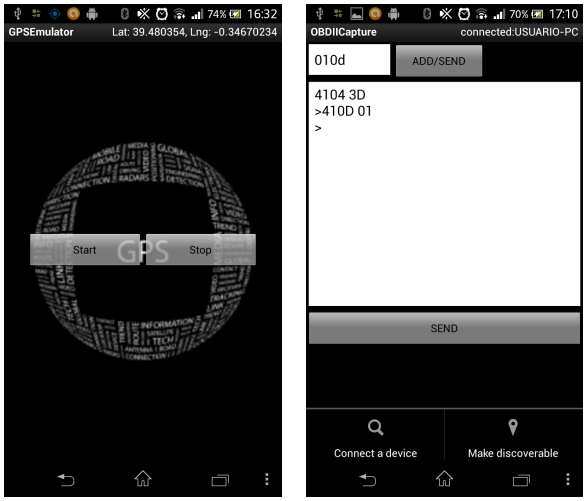
4.2 GPSEmulator

Concerning GPSEmulator, it is used to emulate a (fake) location on Android systems (see Figure 5.a). Basically it creates a service which retrieves simulated geo-positioning data from the VEWE-Server via Bluetooth, and then injects the simulated locations on the Android system; other sources of localization data, such as GPS, WiFi or GSM-based triangulation, are disabled to avoid conflicts. This way the simulated coordinates are assumed as real by other running applications.

The application interface is quite simple. When the service starts, the Bluetooth device connected to VEWE is selected for coordinate retrieval; when the service is stopped, the Bluetooth connection to the VEWE server is closed, and emulated georeferencing ends.

4.3 OBDIICapture

With regard to OBDIICapture (see Figure 5.b), it is a different Android component designed to support the manual introduction of AT commands and OBD-II PID requests, and through which data from a real Bluetooth OBD-II interface is captured. It is useful to analyze the initial message exchange when using real



(a) GPS Emulator. (b) OBDIICapture.

Fig. 5. VEWE Android Components

OBD-II interfaces by providing a log service, and to validate the message output of the VEWE server. In addition, it was also useful to measure the response times and the maximum message rate in a real environment, allowing to tune VEWE so as to achieve a similar behavior.

5 VEWE Implementation Details

VEWE was built using a layered architecture through which the different features of the system become independent, allowing to tweak each of them independently, thereby simplifying the development and maintenance of the system.

The VEWE server is a multithreaded system where separate threads are responsible for handling: (i) data generation, (ii) the graphical environment, and (iii) Bluetooth communications. Communication between threads is achieved through the shared variable paradigm, so that changes introduced by one thread are automatically detected by other threads.

VEWE follows a client-server paradigm, where an Android-based client application queries a Java-based server to retrieve the values of the different parameters stored in the emulated engine ECU (see figure 6). To this purpose it relies on a Bluetooth connection between the client and the server, upon which an OBD-II serial connection is established.

Below we provide details about the vehicle simulator, the emulation of GPS coordinates, and the OBD-II parameter database.

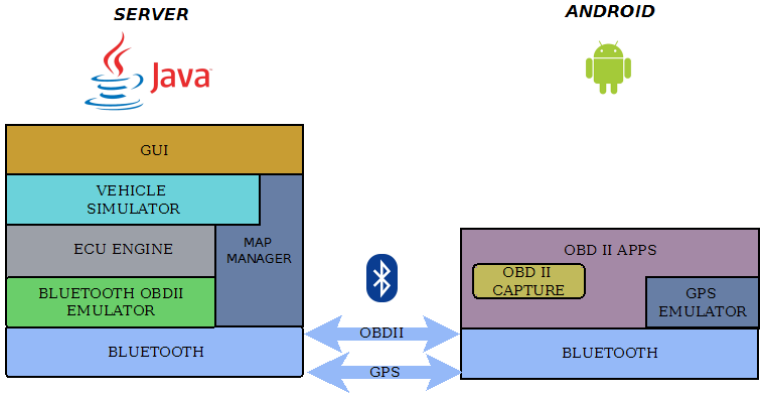


Fig. 6. VEWE architecture

5.1 The Vehicle Simulator

The vehicle simulator is a key element in the VEWE Server, being responsible for modeling vehicle mobility according to the user input, the vehicle characteristics, and the terrain profile. To that purpose it relies on the acceleration calculator module which, by taking into account the different forces affecting the vehicle's mobility - traction, friction, aerodynamic and gravity - determines the acceleration value (see figure 7) using realistic vehicle dynamics. Based on the acceleration value, it then updates the vehicle position on the map, as well as the desired parameters on engine's ECU (e.g. speed, RPM, gear).

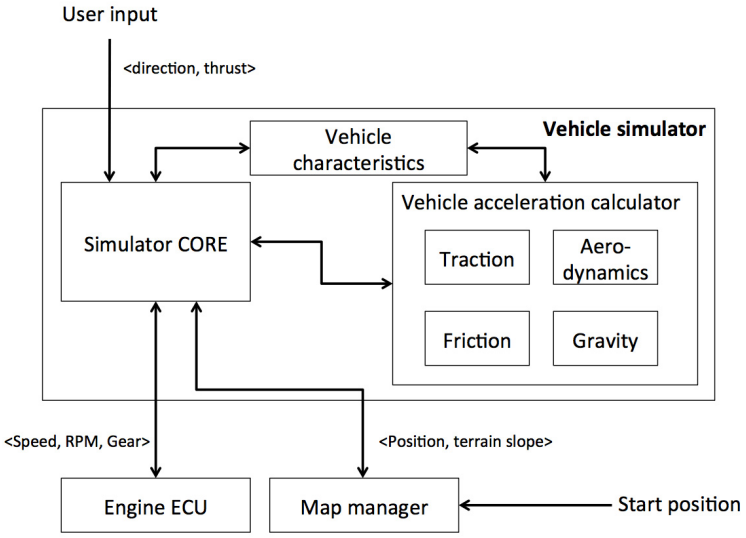


Fig. 7. Overview of the vehicle simulator

Real-time updating of the vehicle position on the map provides the user with a feeling of control over the simulated vehicle using the joystick available on screen. Notice that, by modifying the vehicle characteristics element, different types of vehicles can be modeled.

5.2 Emulating GPS Coordinates

The Map Manager component obtains updated vehicle coordinates from the vehicle simulator described above. In particular, vehicle positions are updated by taking as reference: (i) the current vehicle position, (ii) the orientation, and (iii) the distance traversed; the latter is calculated by combining the vehicle's speed with the inter-sample times.

The Map Manager component is able to provide geopositioning information to external applications through a Bluetooth channel. In our framework, we developed an Android application whose only purpose is to generate fake positions based on the information retrieved from the Map Manager via Bluetooth (see Figure 8). This application, called *GPS Emulator*, uses the *mock location* functionality provided in the Android API to introduce the fake locations into the system; to avoid interferences, it also cancels all other sources of localization information, including GPS, WiFi, or cellular-based positioning. This way, all running applications that register for localization services will receive the mock locations generated by the VEWE Server Map Manager component.

By making the speed calculated using GPS coordinates match the speed value returned by the OBD-II interface, we are able to provide consistent data to application developers.

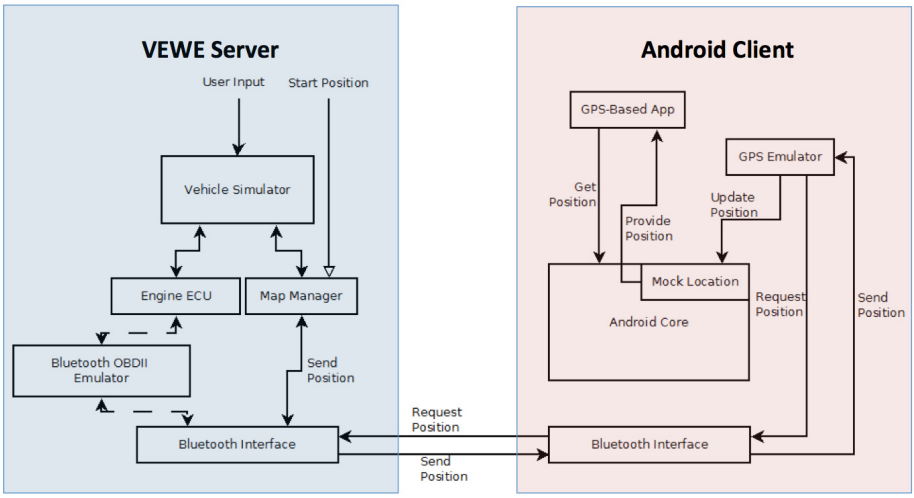


Fig. 8. Emulated position update procedure

5.3 Emulated OBD-II Interface

Providing an emulated OBD-II interface accessible via Bluetooth is one of the main goals of the VEWE platform. The OBD-II interface emulator is able to maintain a serial connection with a client, and adequately process AT commands and PID requests as would occur when using real OBD-II devices.

In order to retrieve the actual PID values, the OBD-II layer communicates with the Engine ECU element, responsible for storing and maintaining all PIDs supported. The Engine ECU handles a data structure (see Figure 9) capable of storing all the vehicle parameters (OBD-II Codes) as well as AT parameters.

The OBD-II interface emulator then converts the retrieved valuers to the appropriate format for delivery through the serial port.

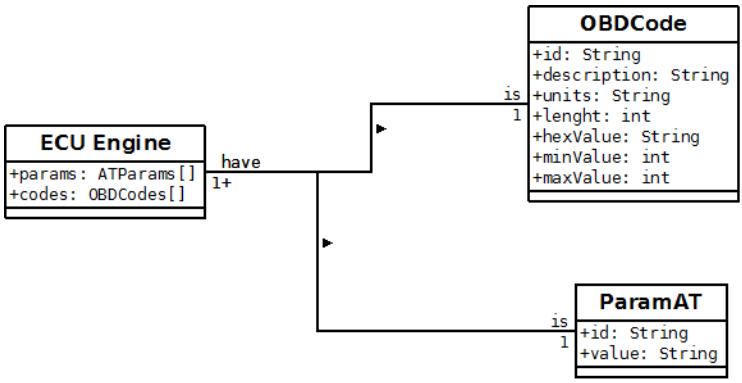


Fig. 9. Structure of the OBD-II parameter database

6 Conclusions

The evolution path towards sophisticated solutions in the ITS sector include smartphone/vehicle integration approaches. Based on current technologies, such approaches require smartphones to communicate with the vehicle's ECU through the OBD-II interface by relying on wireless connectors, typically Bluetooth-based.

In the literature different solutions are starting to emerge that make use of smartphone/vehicle integration using wireless OBD-II interfaces. However, developing such solutions is costly and time consuming, typically requiring several test drives to properly debug and tune the functionality of the applications being developed. In this paper we provide an efficient solution to this problem by introducing a platform able to emulate a Bluetooth-based OBD-II connection, along with the GPS coordinates of the vehicle. To this aim we provide VEWE, which is able to simulate the mobility of a vehicle controlled by a user using realistic acceleration/deceleration dynamics, registering data in a simulated engine ECU accordingly.

Overall, the proposed solution is expected to boost application development in the ITS area, reducing the development effort and costs, and making it feasible for developers without access to vehicles to develop this type of applications as well.

Acknowledgments. This work was partially supported by the *Ministerio de Ciencia e Innovación*, Spain, under Grant TIN2011-27543- C03-01.

References

1. International Organization for Standardization, ISO 14230-1:1999: Road vehicles, Diagnostic systems, Keyword Protocol 2000 (1999)
2. Elm Electronics - Circuits for the Hobbyist, Obd to rs232 interpreter (2013)
3. Martinez, S., Meseguer, J., Zaldivar, J., Calafate, C., Cano, J.-C., Manzoni, P., Fogue, M., Martinez, F.: Smartphones as the keystone for leveraging the diffusion of ITS applications. In: 9th ITS European Congress (2013)
4. Hawkins, I.: Torque: OBD2 Performance and Diagnostics for your Vehicle (2014), `http://torque-bhp.com/`
5. Teng, H.-F., Wang, M.-J., Lin, C.-M.: An implementation of android-based mobile virtual instrument for telematics applications. In: 2nd International Conference on Innovations in Bioinspired Computing and Applications (IBICA), pp. 306–308 (2011)
6. Meseguer, J.E., Calafate, C.T., Cano, J.C., Manzoni, P.: Drivingstyles: A smartphone application to assess driver behavior. In: 2013 IEEE Symposium on Computers and Communications (ISCC), pp. 535–540 (July 2013)
7. Zaldivar, J., Calafate, C., Cano, J.-C., Manzoni, P.: Providing accident detection in vehicular networks through OBD-II devices and Android-based smartphones. In: 2011 IEEE 36th Conference on Local Computer Networks (LCN), pp. 813–819 (2011)
8. Tahat, A., Said, A., Jaouni, F., Qadamani, W.: Android-based universal vehicle diagnostic and tracking system. In: IEEE 16th International Symposium on Consumer Electronics (ISCE), pp. 137–143 (2012)
9. Freematics, Freematics OBD-II Emulator (2013)
10. Briggs, G.: OBDSim (2013)
11. International Organization for Standardization, ISO 15765: Road vehicles, Diagnostics on Controller Area Networks (CAN) (2004)
12. SAE International - Vehicle Architecture For Data Communications Standards, Class B Data Communications Network Interface (2006)
13. International Organization for Standardization, ISO 9141-2:1994/Amd 1:1996 (1996)
14. International Organization for Standardization, ISO 15031-6: Road vehicles – Communication between vehicle and external equipment for emissions-related diagnostics – Part 6: Diagnostic trouble code definitions (2010)
15. The OpenStreetMap Project (2014), `http://www.openstreetmap.org/`
16. Bluecove Team, Bluecove (2008), `http://bluecove.org/`