COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND
INFORMATICS

INTERNET OF THINGS IN AUTOMOTIVE INDUSTRY

Bachelor's thesis

2016 **Dávid Kőszeghy**

COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS

INTERNET OF THINGS IN AUTOMOTIVE INDUSTRY

Bachelor's thesis

Study Programme:	Applied Computer Science
Field of Study:	9.2.9. Applied Informatics
Department:	Department of Applied Informatics
Supervisor:	RNDr. Jozef Šiška, mgr?
Advisor:	Mgr. Anton Pytel

Bratislava 2016 **Dávid Kőszeghy**

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.9. aplikovaná informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský

Názov:

Cieľ:

Anotácia:

Vedúci:
Katedra:
Dátum zadania:

Dátum schválenia:

.....
študent

.....
vedúci práce

Declaration of Authorship

I confirm that this Bachelor's thesis is my own work and I have documented all sources and material used.

Bratislava May 26, 2015 .....

Acknowledgements

I would like to thank my supervisor...

Abstrakt

Tu je text slovenskej verzie abstraktu

Kľúčové slová: *slovo1, slovo2, slovo3, slovo4*

Abstract

In this bachelor thesis we will show you the possibilities of connected cars, options in what advanced services a connected vehicle can provide and how you can do it in the first place. After a thoughtful study of our work anybody with computer science background will be able to build a device which will be capable of advandced services based on his/her needs.

Keywords: *Automotive industry, IoT, Big Data, Embedded device*

Contents

Introduction

Most of us who watched Knight Rider as a kid expected that by 2015 we would be driving self-aware cars like KITT - cars that would drive us without problems from point A to point B while entertaining us. However we can resort to that this is only partially true, we have successful tries at completely autonomous cars(e.g Google's car) but for ordinary customer they are not available. What is available is connected cars. Connected cars provide the possibility of internet-based transfer of information. This can be obtained either with embedded devices in car or smartphone. Multiple car manufacturers sell connected cars where different OEM(Original Equipment Manufacturer) devices or applications. These OEMs offer various services which are provided to you when you purchase their car, but there is a small problem. Often these OEM devices have been designed to work with applications provided from car manufacturer and it is not possible to use other applications to gather information from the car. You can not configrue them to the extent as smartphones have apps. It is either you will get used to the OEM solution or go and buy whole solution from somebody else. It is like we are stuck in the 90s, where software offered to you had come on this particular hardware and you could not choose otherwise. Imagine you would buy smartphone with predifined set of applications and you could not change them or add new. To sum it up, new connected cars lack easy configuration of information gathering, using this information according to wishes of customer not car manufacturer. So in our thesis we will provide solution for these problems, build a device which will be capable of collecting information and act based on it. By following our thesis insight will be gained into how to connect car and what possible services might be obtained. In first chapter we will establish some common ground on upon which we can build foundations of this thesis. In next step we will describe the specification of problem, design a solution for it, and show implementation of the solution to this given task.

1. Overview

1.1 Solutions available

In this section we will provide an overview of existing solutions which are similiar with the goal of this bachelor thesis.

1.1.1 Automatic

“The Automatic car adapter plugs into just about any car’s standard diagnostics (OBD-II) port. It unlocks the data in your car’s on-board computer and connects it to your phone via Bluetooth wireless.”[5]
Automatic turns almost any car into a connected car. By pairing Automatic’s adapter and apps for iPhone, Android, and web, drivers are able to enhance their driving experience with a host of connected services on the Automatic platform. Automatic helps to diagnose engine trouble, also provides acelerometer which measures car orientation to detect serious collision. By using audio signals Automatic is able to provide feedback to the driver. As Automatic connection is wireless, all data is encryptet by using 128-bit AES encryption.[2] More detailed hardware specification can be found in the attachment chapter[5]. This particular solution is proprietary and can be bought in the U.S.

1.1.2 VI Monitor

The VI Monitor works by reading the data stream straight form vehicle electronic control unit via OBD-II. The VI Monitor has also a 3.5" touch screen display and G-sensor. It has an advanced diagnostics tool with the ability to view and reset engine fault codes and comes with sophisticated comparision software. Main difference comparing Automatic and VI Monitor is in set of tools, VI Monitor allows for precision braking and acceleraion tests by using accelerometer and car data.[26] For more detailed overview of capabilities of VI Monitor consult [5]

1.2 Internet of Things

The Internet of Things (IoT) is a global infrastructure concept to further informatization of society, enabling advanced services by interconnecting things based on available technologies.
The IoT technology aims to build a set of networks in which each object is connected. In the IoT, all objects has the storage and computing power.[23] Through identification, data capture, communication and processing capabilities, the IoT makes full use of things to offer services to all kinds of applications.
Whilst IoT is a hot topic in the industry it is not a new concept. It was initially put forward by Mark Weiser in the early 1990s. This concept is opening up huge opportunities for both the society and individuals. However, it also involves risks and undoubtedly represents an immense technical and social challenge.[8]

1.2.1 Concept

"Things", in IoT refer to a various devices. From hardware level they are all designed to do different things, collect data from various enviroments and sources. Howewer, in software aspect they behave more generally, in simplistic form it is a thing which can report data and act upon them.
Objectification is important, because then it can be combined with many things to work together and communicate between them. Current market example could be the smart thermostat systems combining washer/dryers that use Wi-Fi for remote access.

1.2.2 Benefits

IoT is generating a lot of interest in a wide range of industries. Here are a few examples of some significant early adopters:

- In the healthcare field, medical device manufacturer Varian Medical Systems is seeing a 50 percent reduction in mean time to repair their connected devices.[22] With IoT, Varian reduced customer service costs by $2,000 for each problem resolved remotely, with 20 percent fewer technician dispatches worldwide.
- Italian tire maker Pirelli is using IoT to gain valuable insights about the performance of its products in near real time.[20] The company is using an analytics platform to

manage the huge amounts of data gathered directly from sensors embedded in the tires in its Cyber Tyre range. The system allows the pressure, temperature, and mileage of each tire to be monitored remotely. By keeping these factors in range, fleet managers can have a significant impact on fuel economy and safety. In a trial covering nearly 10 million miles, Cyber Tyres saved the equivalent of $1,500 per truck per year[4].

- Ford Motor Company's Connected Car Dashboards program collects and analyzes data from vehicles in order to gain insights about driving patterns and vehicle performance. The data is analyzed and then visualized graphically using a big data platform. Among the goals are better vehicle design and improved safety for occupants.[20]

- US-based HydroPoint Data Systems is using the loT in its WeatherTRAK application to enable landscape irrigation sys-tems, sensors, and computers to communicate autonomously online to identify leaks and other potential system problems. [20]

IoT has the potential to transform the way companies make products, track goods and assets in the supply chain, monitor the performance of systems in the field, provide security for employees and facilities, and provide services to customers. Clearly, it's enabling transformation in both the private and public sectors.
"I firmly believe that there is not a single industry that won't benefit from IoT." – said Vernon Turner, Senior Vice President of Reasearch and IoT Executive Lead at International Data Corp.(IDC). IoT is also changing the way businesses impact society and the environment. "Overall, the world needs better and more sustainable ways to live," said Stephen Miles, research affiliate at the Center for Biomedical Innovation at the Massachusetts Institute of Technology. "To accomplish this, companies need better, more holistic models that capture a complete picture of what is happening so that they can better access and optimize these systems."

1.3 I^{2}C

I^{2}C protocol is a very popular serial protocol, mainly because it is cheap, simple and allows to connect multiple devices. Whole I^{2}C bus can be made of only 2 lines: SDA and SCL. SDA is a data line which is carrying data from one device to another and SCL is clock line which main purpose is to synchronize all the devices connected using the I^{2}C bus. The SDA and the SCL lines are mandatory for all devices which support connection using I^{2}C bus[13]. The block diagram representation of an I^{2}C bus is as shown in 1.1

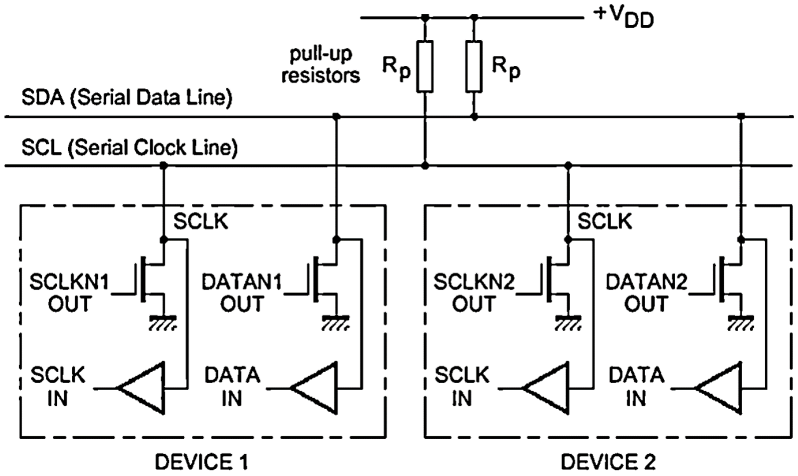


Figure 1.1: I^{2}C bus.

1.3.1 Bus signals

Both signals (SCL and SDA) are bidirectional. They are connected through resistors to a positive power supply voltage. This means both lines are high when the bus is free. Activating the line means pulling it down (wired AND). The number of the devices on a single bus is almost unlimited – the only requirement is that the bus capacitance does not exceed 400 pF. Because logical 1 level depends on the supply voltage, there is no standard bus voltage.[12]

1.3.2 Hierarchy

I^{2}C devices are can be registered as master or slave. Masters initiate a message and slaves respond to a message. A master can have multiple slaves and any device can be master-only, slave-only, or switch between as provided in image 1.2.

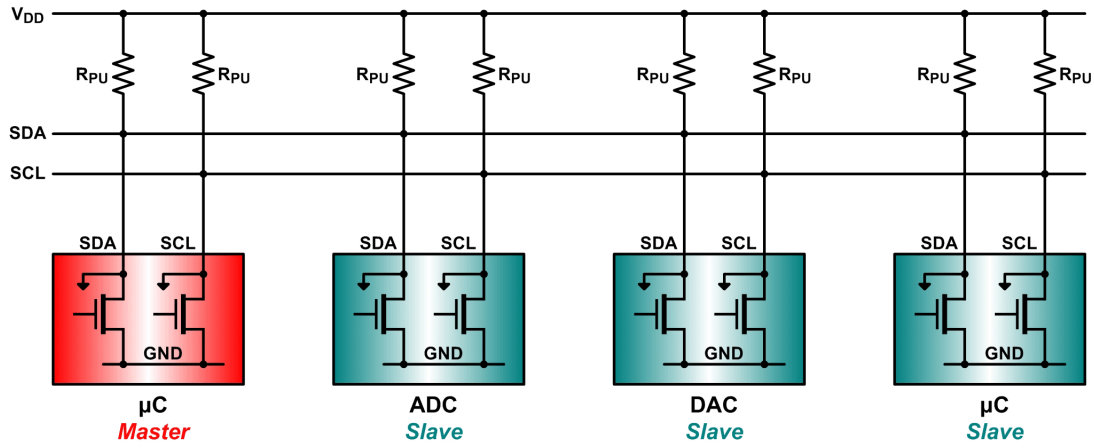


Figure 1.2: Multiple I^2C devices connected on the bus.

1.4 Single-board computer

Sensor monitoring systems are used in many applications such as greenhouses, ecology studies, automobiles, robots, and medical devices. These systems are essentially used to study environment data like temperature, light, humidity, or air pressure. These sensor measures are used in manual and automation control based systems[1]. The SBC is complete functional computer built on single circuit board. It has microprocessor, memory, input/output and other features depending on the model and manufacturer. SBC are used for educational purposes, embedded solutions and development research/systems.

1.4.1 Platforms

For making a informed decision which SBC to use for this thesis, we provide short overview of choosen manufacturers and their platforms for embedded devices.

1.4.1.1 Raspberry Pi

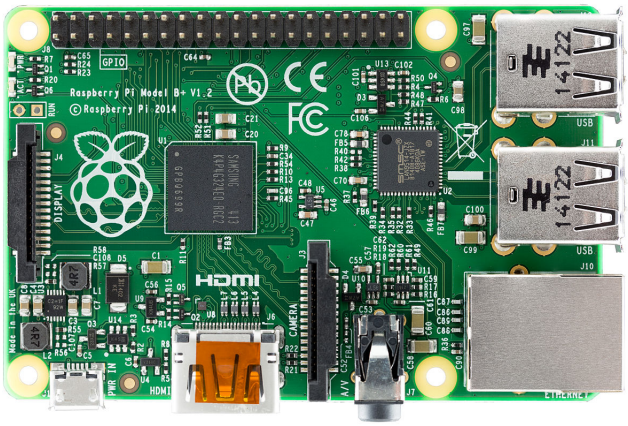


Figure 1.3: Rasberry Pi Model B+ v1.2 by Lucasbosch

Raspberry Pi is a credit-card sized mini computer as shown at Figure 1.3. It is a low cost solution to many projects, and its main goal is to enable people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. It is fully capable of running operating system based on ARM architecture. It has been used in a wide array of digital maker projects, from music machines and parent detectors to weather stations and tweeting birdhouses with infra-red cameras.[11]. Raspberry Pi foundation develped various models with similiar hardware specifications. At the time of writing this thesis the most advanced model was Raspberry Pi 3. Hardware specifications are shown at Table 1.1. Best addition to previous version was on-board wireless chip, which allowed the device to be connected to the wireless network.

SoC:	Broadcom BCM2837
CPU:	4x ARM Cortex-A53, 1.2GHz
GPU:	Broadcom VideoCore IV
RAM:	1GB LPDDR2 (900 MHz)
Networking:	10/100 Ethernet, 2.4GHz 802.11n wireless
Bluetooth:	Bluetooth 4.1 Classic, Bluetooth Low Energy
Storage:	microSD
GPIO:	40-pin header, populated
Ports:	HDMI, 3.5mm analogue audio-video jack, 4x USB 2.0
	Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)

Table 1.1: Raspberry 3: Hardware specification

1.4.1.2 The Beagles

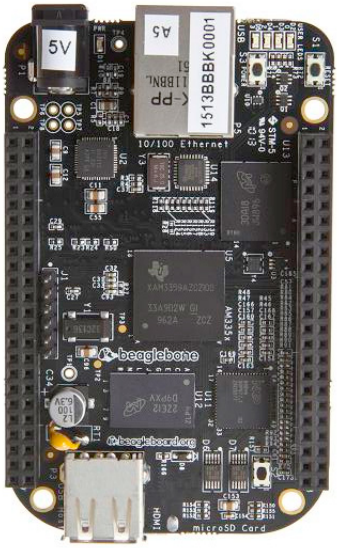


Figure 1.4: BeagleBone Black

The BeagleBoard Foundation is a US-based non-profit corporation existing to provide education in and promotion of the design and use of open-source software and hardware in embedded computing. Providing a platform for the owners and developers of open-source software and hardware to exchange ideas, knowledge and experience.[17]. The Beagles are open-hardware and open-software computers, that can be used for numerous projects, same as Raspberry Pi. In time of writing this thesis most popular model among the Beagles was BeagleBone Black shown at Figure and detailed hardware specification at Table 1.2.

SoC:	Broadcom BCM2837
CPU:	AM335x 1GHz ARM® Cortex-A8
GPU:	SGX530 3D
RAM:	512MB DDR3 RAM
Memory:	4GB 8-bit eMMC on-board flash storage
Networking:	10/100 Ethernet
Storage:	microSD
GPIO:	2x 46 pin headers
Ports:	HDMI, Ethernet, LCD, GPMC,MMC1/2, 4 Serial Ports, CAN0

Table 1.2: BeagleBone Black: Hardware specification

As we can see in comparison to Raspberry Pi, BeagleBone is a SBC more oriented for controlling low-level application. However BeagleBone lacks any USB ports for connecting any peripherals, so to extend BeagleBone functionality one must buy expansion boars.

Which is not necessarily bad, but they might not be available in common as classic USB peripherals.

1.4.1.3 The Intel® Edison

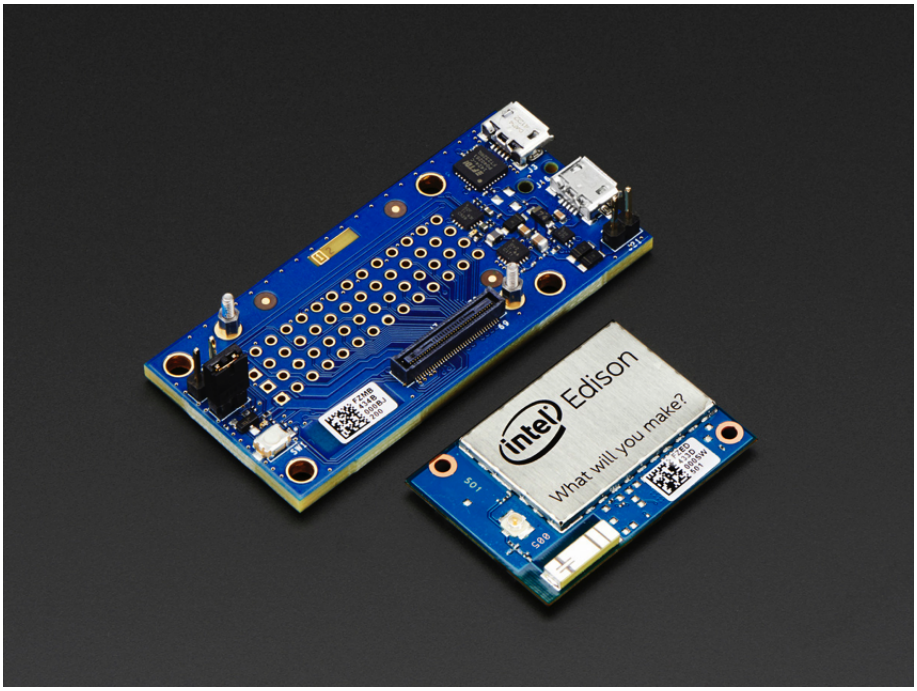


Figure 1.5: Intel® Edison with Breakout Board

The Intel® Edison development platform is designed to lower the barriers to entry for a range of Inventors, Entrepreneurs and consumer product designers to rapidly prototype and produce IoT and wearable computing products.[7]
Intel® Edison is very small SBC which provides very interesting hardware specification, which can be found in Table 1.3. However to connect anything to Intel® Edison, expansion board is needed because I/O pins of Intel® Edison are grouped in very small 70 PIN I/O Connector. One example of expansion board is Edison Breakout board which has a minimalistic set of features and is slightly larger than the Edison module as shown at Figure 1.5.
This is concept which is different from the Raspberry and the Beagles. If a project needs different set of capabilities instead of changing whole SBC, it is enough to change just the expansion board. One main disadvantage with starting with this platform is higher cost of obtaining Intel® Edison and exapnsion board.

SoC:	22-nm Intel® SoC
CPU:	dual-core, dualthreaded Intel® AtomTM CPU at 500Mhz and a 32-bit Intel® QuarkTM microcontroller at 100 MHz
GPU:	SGX530 3D
RAM:	1 GB LPDDR3 POP memory
Memory:	4GB eMMC
Wireless:	Broadcom* 43340 802.11 a/b/g/n; Dual-band (2.4 and 5 GHz) On board antenna or external antenna
Bluetooth:	Bluetooth BT 4.0
Storage:	microSD
GPIO:	70 pin headers, populated
Ports:	USB OTG, 2 Serial Ports

Table 1.3: Intel® Edison: Hardware specification

1.4.2 Peripherals

To provide neccessary capabilities to solve this bachelor thesis, we might need peripheral devices to extend capabilities of developed embedded device.

1.4.2.1 Bluetooth / WiFi Combination USB Dongle

Figure 1.6: Bluetooth / WiFi Combination USB Dongle

If SBC choosen to complete this bachelor thesis would not have WiFi/Bluetooth capabilities, this adapter in Figure 1.6 will be used to provide it. Technical details can be found in Chapter 5. Advantage of this USB dongle is that it 'saves' one USB slot by integrating WiFi and Bluetooth technology into one device.

1.4.2.2 Adafruit 10-DOF

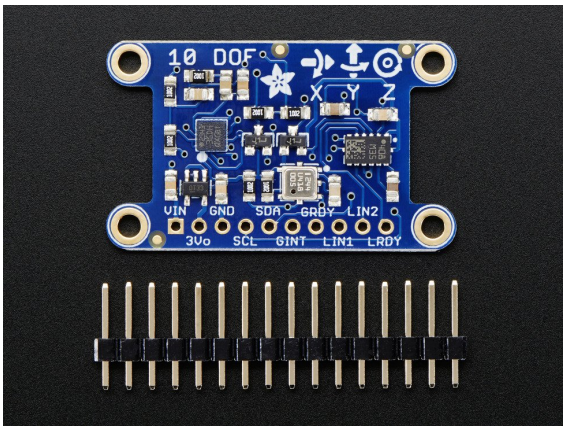


Figure 1.7: Bluetooth / WiFi Combination USB Dongle

This inertial-measurement-unit(Figure 1.7) combines 3 sensors to give you 11 axes of data: 3 axes of accelerometer data, 3 axes gyroscopic, 3 axes magnetic (compass), barometric pressure/altitude and temperature. Since all of them use I2C, you can communicate with all of them using only two wires. Most will be pretty happy with just the plain I2C interfacing, but we also break out the data ready and interrupt pins, so advanced users can interface with if they choose.[14].

1.4.2.3 Adafruit FONA 3G Cellular + GPS

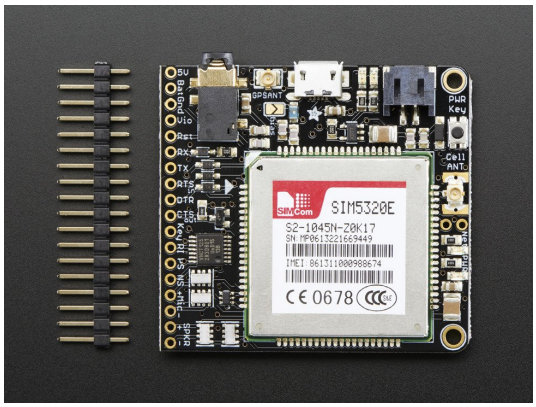


Figure 1.8: Adafruit FONA 3G Cellular Breakout - European version

The FONA 3G has better coverage, GSM backwards-compatibility and even sports a built-in GPS module for geolocation and asset tracking. This all-in-one cellular phone module with that lets you add location-tracking, voice, text, SMS and data to your project in a single breakout as shown in Figure 1.8.[15]

1.5 Automotive electronic systems

As we will be gathering data from electronic systems of the car, lets overview how they work, what systems and protocols are present in modern cars and how to use them to our benefit.

1.5.1 Perspective

To gain better perspective into how complex elextronic systems in cars are, lets go back a little into history. In 2002, Leen and Heffernan said in their work "Expanding automotive electronic systems":

> The growth of electronic systems has had implications for vehicle engineering. For example, today's high-end vehicles may have more than 4 kilometers of wiring-compared to 45 meters in vehicles manufactured in 1955. In July 1969, Apollo 11 employed a little more than 150 Kbytes of onboard memory to go to the moon and back. Just 30 years later, a family car might use 500 Kbytes to keep the CD player from skipping tracks. ([18])

It is now 2016 and electronic systems evolved so much that you can find 100 milion lines of code in software of average modern high-end car[6]. Modern cars nowdays have more than 30-50 Electronic Control Units(ECUs). At Figure 1.9 we can see network of electronic components inside a car.

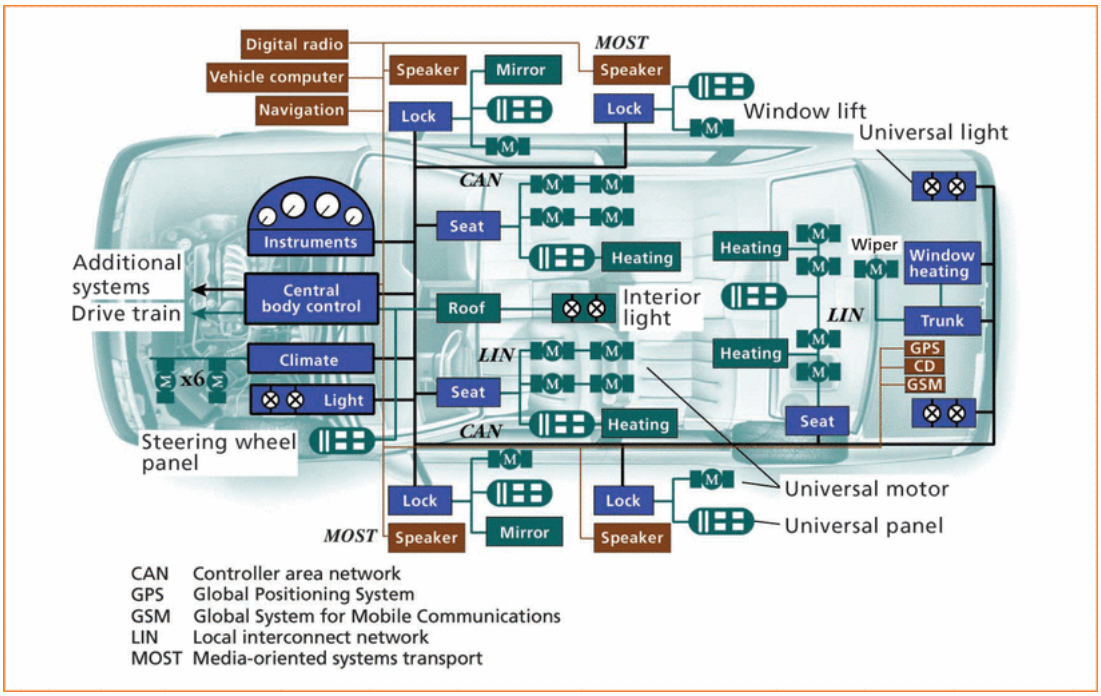


Figure 1.9: Vehicles electronic network

1.5.1.1 In-vehicle networks

Just as LAN or Wi-Fi connects computers, control networks connects a car's electronic equipment. By interconnecting different pars of ECUs, vehicle is able to share information amongst all distributed applications. In the history of automotive industry, wiring was just connecting element A to element B, however when electronic content in cars increased, the use of more and more cables to link elements growed enormously. To provide an example, Motorola reported in a 1998 press release, that replacing wiring cables with LANs in the four doors of a BMW car reduced the weight by 15 kilograms. Beginning in the early 1980s, centralized and then distributed networks have replaced point-to-point wiring.[19]

1.5.2 Controller Area Network

The Controller Area Network(CAN) is an International Standardization Organization (ISO) defined serial communications bus specifically designed with real-time requirements in mind. Developed in the 1980s by Robert Bosch, its ease of use and low cost has led to its wide adoption throughout the automotive and automation industries[10]. It is currently the most widely used vehicular network[18]. CAN is designed for signaling rates up to 1 Mbps, high protection against electrical interference, and an ability to find and repair data errors.

The CAN communications protocols, ISO 11898, conforms to the Open Systems Interconnection (OSI) model that is defined in terms of layers. Protocols specify how data travel between devices on the network. Physical layer defines how to communicate between devices which are connected. Lowest two layers of the ISO/OSI model are defined as the data-link and physical layer shown in Figure 1.10.

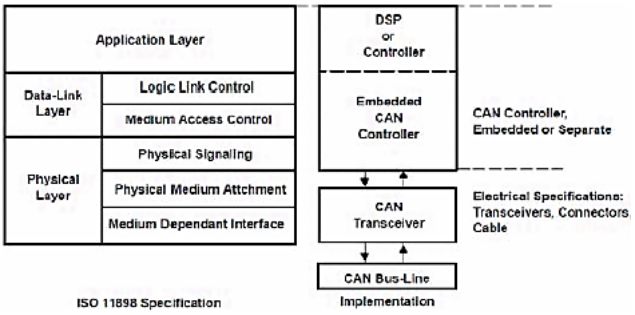


Figure 1.10: Layers of CAN Communications protocol, ISO 11898

When data is transmitted over a CAN network no individual devices are addressed. Instead, all messages have an unique identifier, a unique tag of its data content. Not only it identifies the message contents, but also the message priority as how much important data contents are. Lower the message priority, the more important message and it is delivered in prior to

messages with higher priority. When a device connected to the network wants to transmit message, it passes the data and the identifier to the CAN controller with a relevant request to transmit information. CAN controller formats the message conents and transmits the data in form of a CAN frame as shown in Figure 1.11.

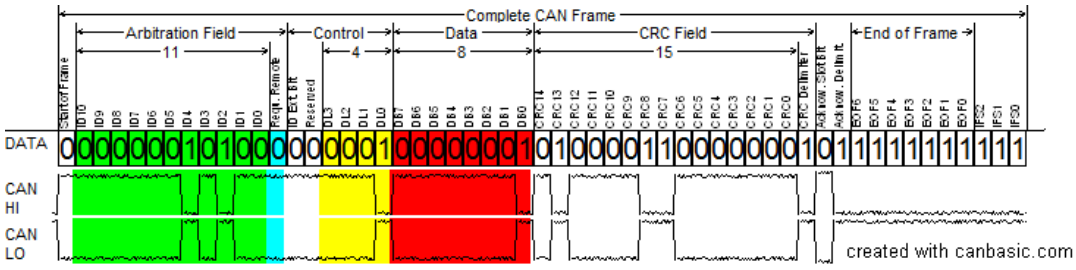


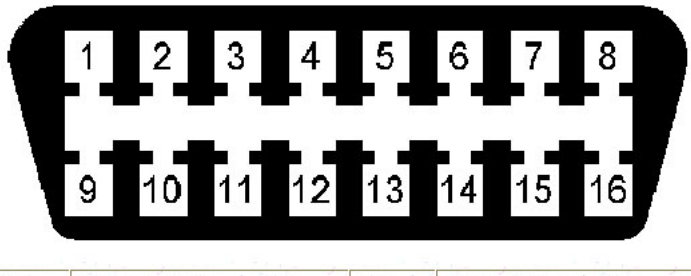
Figure 1.11: CAN bus frame, by Erniotti (output of slef programmed software) [GFDL (http://www.gnu.org/copyleft/fdl.html) or CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

1.5.3 Conclusion

By understanding standard messaging protocol used in cars, we are able to read and create messages. Creating and reading messages relies on correct construction of data payload and assignment of correct identifier to the message, however specification of how to create this particular parts of the message is not publicly available, moreover specification is proprietary and probably different for every car manafacturer. To discover what data in messages are and what identifier are used, data analysis of closed data dumps from different states of the car(Ignition on, windows down, engine off, etc.) can be used, however this method is dependant on concrete make and model of the car, and is not suitable for large scale applications or general use.

1.6 OBD-II

When environmental pollution became a world issue in the 1970s, the U.S. created the Environmental Protection Agency (EPA). EPA established a new standard to limit the environmental pollutants emitted from vehicles. The car makers devised an electronic control system for the fuel supply and ignition devices, based on the standard. In addition, the Society of Automotive Engineers (SAE) established OBD in 1988 as a standard for the plug connector and on-board diagnosis system[16] as shown in Figure 1.12.



PIN	DESCRIPTION	PIN	DESCRIPTION
1	Vendor Option	9	Vendor Option
2	J1850 Bus +	10	j1850 BUS
3	Vendor Option	11	Vendor Option
4	Chassis Ground	12	Vendor Option
5	Signal Ground	13	Vendor Option
6	CAN (J-2234) High	14	CAN (J-2234) Low
7	ISO 9141-2 K-Line	15	ISO 9141-2 Low
8	Vendor Option	16	Battery Power

OBD-II Connector and Pinout

Figure 1.12: Pinout of the plug connector

OBD standard developed over-time and in time of writing this thesis OBD-II is standard which is used in the industry.

1.6.1 Standard

The OBD-II standard specifies the type of the connector and its pinout, the messaging format and electrical signalling protocols available. Difficulties may arise because of the deficiency in the OEM specific implementation of the standard. According to The OBD Home Page: There are five basic OBD protocols in use, each with minor variations on the communication pattern between the on-board diagnostic computer and the scanner console or tool. Formula CAN is the newest protocol added to the OBD specification, and it is mandated for all 2008 and newer model years[25]. OEM provides a list of vehicle queries which can be send to the OBD-II port for diagnostics and gathering real-time parameters, but the list may vary from car to car. Each of this queries has a specified response on how to decode returned data again by OEM and may vary from another manufacturer respones. Finally, the OBD-II standard provides an extensible list of DTCs(Diagnostic trouble codes). As a result of this standardization, a single device can query the on-board computer(s) in any vehicle [24], however this is again based solely on OEM so for getting all posible informations from the OBD-II port, specifiaction from OEM is needed.

1.6.2 Communication

As mention above to access information from sensors attached to a vehicle via a OBD-II port, special codes called Parameter IDs(PIDs) are used. Typically a process of retrieving data from vehicle trough OBD-II port is as follows.

1. Requested PID is entered either through connected computer or scan tool.
2. Request is send to vehicles CAN bus.
3. Device which is responsible for given PID recognizes it and then reports the value back to the bus
4. Connected computer or scan tool reads the response.

Figure 1.13: Query OBD-II process

From Figure 1.13 we can see that OBD-II port is in simple terms subset of CAN frames. Identifier and data payload is supplemented from OBD-II protocol based on concrete PID query.

1.6.3 Conlusion

Main advantage of OBD-II port is that obtaing a single parameter from the car is relatively simple. It is enough to know correct PID to query the OBD-II port and the encoded response is the wanted information but it is not suitable for real-time telemetry, because of repetiton rate of the available data. The more different data is read out, the lower repetition time will be for each queried type, due to the fact that information is only accessible by question answer basis.

1.7 Accelerometer

An accelerometer measures proper acceleration which is the acceleration it experiences relative to freefall, and is the acceleration that is felt by people and objects. Put another way, at any point in spacetime the equivalence principle guarantees the existence of a local inertial frame, and an accelerometer measures the acceleration relative to that frame.[9]As a consequence an accelerometer at rest relative to the Earth's surface will indicate approximately 1 g upwards, because any point on the earth's surface is accelerating upwards relative to a local inertial frame. To obtain the acceleration due to motion with

respect to the earth, this "gravity offset" should be subtracted.
The reason for the appearance of a gravitational offset is Einstein's equivalence principle[21], which states that the effects of gravity on an object are indistinguishable from acceleration of the reference frame. When held fixed in a gravitational field by, for example, applying a ground reaction force or an equivalent upward thrust, the reference frame for an accelerometer (its own casing) accelerates upwards with respect to a free-falling reference frame. The effect of this reference frame acceleration is indistinguishable from any other acceleration experienced by the instrument. An accelerometer will read zero during free fall. This includes use in a spaceship orbiting earth, but not a (non-free) fall with air resistance where drag forces reduce the acceleration until terminal velocity is reached, at which point the device would once again indicate 1 g acceleration upwards.
For accident detection we will use accelerometer to determine if accident occured[3].

2. Specification

2.1 Requirements

2.1.1 Hardware

Provide an embedded device which will be running operating system of your preference. Design how to connect all peripheral devices.Setup the device with applications to operate properly such as Internet, Wi-Fi, GPS and additional sensors. Figure out how to connect your device to the vehicle's OBD-II service port. Assamble and provide this device as a hardware part of this bachelor thesis.

2.1.2 System

Configure the system to every sensoric input so you can communicate with them. Use any protocol for communication which will suit your solution. Develop a application for collecting information from various sources, design the system to be easily configurable, without needing to change large portions of the system. System will have modular capabilities of connecting various sources of information. Develop a routing which will enable to configure which way data flows from input sources to output sinks. Design the application for modular adding of different outputs. Define a format in which are information passed from your server. Create an interface where user can configure this input functions to increase adaptivity of your system. Figure out how to send data to the server connected to the Internet. For accessing through WiFi, create RESTful web application which will user to access your web app. In web application add a configuration webpage where user can easily configure your system. Parameters of modules connected to your application will have to be configurable by the user, define them and provide a configuration method.

2.1.3 Application

Provide examples which will utilize your modular application, gathering information collected from connected vehilce. The examples needs to preform logic deduction from accessed data and act upon it.

2.1.4 Output

Output of your bachelor thesis should be application which upon correct instalation on embedded device is capable of monitoring various inputs of sensoric data, based upon configuration distribute and make them accessible for different services.

3. Solution

In this chapter of bachelor thesis I will provide a solution of my bachelor thesis problem.

3.1 Hardware

For the start we need device which will be capable of running operating system. There are many various platforms which provide single board computers, and after considering pros and cons I have choosen raspberry platform as most suitable becouse is is capable of running full linux operating system, have 4 USB slots which will be needed for peripheral devices and foremost there is a pleathora of hardware addons, sensors and boards which works with raspberry models, not to mention availablilty of information how to connect them, tutorials how to setup them. In time of writing this thesis I can say raspberry platform is the most popular platform for single board computers.

3.1.1 Device

Since we will be building embedded device which will allow us to gather data and send them over to the Internet, it is necessary to choose a single-board computer which provides means for connecting to the Internet either by connectig using built-in hardware or peripheral device. For connecting various sensors SBC must provide enough GPIO pins and support communication protocols commonly used with sensors e.g I^2C. After picking the right platform we need to pick model. Newest model on the market from raspberry platform in time of developing this thesis was Raspberry Pi Zero which was not suitable for the needs of this thesis becouse it lacks usb ports. It has support for On-the-go USB cables but we need device which supports atleast 2 or 3 usb ports. Which we would need to solve by buying USB hubs. The best suitable device was Raspberry Pi 2, which has 4 USB ports, 40 GPIO pins and micro SD card slot. This device covers almost all the needs of this thesis. We will need to extend it with WiFi/Bluetooth USB dongle which will enable us to create WiFi access point. In time of writing this thesis Raspberry Foundation released new device Raspberry Pi 3 which is even more suitable for us because it has buitl-in Wireless LAN. So my recommendation for the reader is to go with Raspberry Pi 3 model.

3.1.2 Operating System

The most suitable operating system for our needs is Linux based, because it is open-source. Concrete distribution of Linux based operating system will be Arch Linux. The main gain of using Arch Linux is it is extremely lightweight distribution with KISS principle in mind also it is a rolling release which means it is always always up-to-date. Port of Arch Linux distribution is Arch Linux ARM which carries forward the Arch Linux philosophy of simplicity and user-centrism, targeting and accommodating competent Linux users by giving them complete control and responsibility over the system. Instructions are provided to assist in navigating the nuances of installation on the various ARM platforms; however, the system itself will offer little assistance to the user[CITATION NEEDED]. Installing and basic configuration of Arch Linux will not be covered in this thesis, as I would repeat myself. Arch Linux has one of the best documentation online, so if you are not familiar with how to install and configure this particular Linux distribution, follow documentation from Arch Linux website.

3.2 Configuration

In this section I will cover configuration of various applications and system properties to set up our device properly for our needs, howewer it will not be complete configuration, as peripherals connected to this device will also need some degree of setup and configuration I will cover that in their respective sections.

3.2.1 Database

For storing information on developing device, it needs to be capable of running database. As it is with linux there, are a lot of databases to choose from and to suit our needs we need to vary on one fact, and that is where all gathered information will be stored. Raspberry can support very large memory cards, up to 512 gigabytes, and with various read/write speeds, so one could find card which will suit oneself needs.[Citation needed] In spite of availability of different micro SD cards, everyone has limited read/write cycles after which card become unreliable. Another solution is not to store information on card at all. Typical linux database will create database file to which it stores data, in configurating we could point database to create this file in a special directory. On Linux it is called /tmp[citation needed], this direcory is allocated not on memory card but instead on RAM memory, which would save us need of large and expensive card. It comes with disadvantage, after restart or shutdown of the device, we would not be able to access stored data as RAM memory is votelatile[citation needed]. My solution for database is to use special database which sets the database in RAM memory but it is configurable to save snapshots(copies) to the micro SD card. Database is called Redis. Redis is an open source, in-memory data structure store, used as database, cache and message broker.[citation needed] Sample configuration on which database runs on our device will be provided with other configurations on GitHub repository under name “redis.conf” in folder “confs”.

3.2.2 Wireless Access Point

To automaticly create wireless access point when the device boots up we need several configrations and applications. Firstly the compatibility of device. To create software access point you will need nl80211 compatible wireless device[Citation needed], which support AP operating mode. Application which will be needed is hostapd, iw and dhcpcd. In the confs folder on github you can find configuration file for hostapd which needs to be copied into `/etc/hostapd/`, and a systemctl rule, which needs to be copied

into `/lib/systemd/system/`. After that reload systemctl dameon, start and enable wifi-ap.service. If you have compliant device it will automatically create wireless Access Point which is defined in hostapd.conf.

3.2.3 Internet

As I am using Adafruit FONA 3G Cellular + GPS modem, creating connection to the Internet is possible in different ways. First is using pppd daemon which is capable upon configuration to create ppp0 device which will be used to connect to the internet. Configuration files for ppp can be found again on GitHub repository in `conf/ppp` folder. Another approach is with wvdial, which congiguration can also be found in repository. At last FONA 3G is supported by QMI modem protocol. For dialing with qmi one needs to install libqmi and net-tools packages. You will need to know to which apn you need to connect from your mobile internet service provider. After that replace APN in qmi-network.conf from config folder at GitHub, copy it to `/etc/`. From script folder found in thesis repository copy a helper script, and then simple `qmi_setup.sh start` starts the connection to the internet. If your SIM card is PIN locked you will need to unlock it manually trough AT command or with help from `mmcli` command.

3.3 Peripherals

For providing proof of concept device which is capable of gathering information from various sources I will connect to the device numerous peripherals which will generate information. In upcoming subsections I will describe them and provide configuration instructions if necessary. I believe that connecting this peripherals is such trivia that it does not need to be explained to the reader, if however someone would not have sufficient knowledge of how to connect these to the raspberry pi I suggest to find some guide on the internet for correct connections to be made.

3.3.1 Adafruit 10-DOF

As for configuration you will need packages `i2c-tools` and `lm_sensors`, then get config config.txt from thesis reposirtory and copy it to /boot/config.txt, enable mondules `i2c-dev, i2c-bcm2708` in `/etc/modules-load.d/raspberrypi.conf`. Thats it, now with command `i2cdetect -y 0` is should work. if not follow troubleshoot documentation on raspberry pi on archlinux wiki.

3.3.2 Adafruit FONA 3G Cellular + GPS

As for configuration there are multiple ways how to connect to the internet, send SMS and make voice calls, all of them boils down to using right AT commands through serial console which is automatically created by the kernel.After connecting modem through usb to the raspberry, kernel will create 4 serial ports(debug,nmea,serial,modem). After starting gps chip with correct command `at+cgps=1` at serial port nmea will start to push gps data on nmea port at 115200 baud rate.

3.4 Software

3.4.1 Problem abstraction

To solve given task, lets firts look at data gathering and what can be done to optimize proccess of collecting infromation from often vary different sources. For correct conclusion to be made, we need to abstract the meaning of car connection as merely only as one of infromation sources. To solve how to easily interconnect different input sources(information producing) to even more different output sinks(information sending), lets develop a framework. A data applicaton module which will provide easy connecting of different sources with data producing modules to any data accepting modules. So to begin lets define some construction blocks of this framework so we can easily build a suitable application.

3.4.2 Data Logger framework

This framework will be a standalone module which will be incorporated in the final application. Main goal of this framework is to provide easy way how to setup communication between various inputs(sensoric data, IoT devices, cars,etc) and outputs(database,application, server, cloud,etc). Framework is structured by main app, modules and routes. Data Logger is configurable trough internal function and by file which serves for storing configurations of each model and routes. This is the first version of the framework and will probably undergo some more development.

3.4.2.1 Module

Module is basic structure for every item. It is a encapsulating class for every information creating or sending application/module. By this framework is able to objectify every application be it different, and extract data from generating one or send data to to accepting one. By this single feature we will be albe to get data from OBD port of the car system the same way as from accelometer present at Adafruit 10 DOF sensoric board. This is achieved by writing specific modules to parse/get data from the source, which are then run under Data Logger. Every module can be generating data, or accepting data and acting upon it. For modul to be generating or accepting one, module must be implemented by the same interface so it can be easily handled by Data Logger, otherwise this would not be much of a framework if for every module there would need to be specific functions to run to obtain or accept data. Configurating modules are easy as from framework we can pass settings for every module separatly, either from application or much easier from a file. This configuration file has a

specific block of configruation settings for every module present in current implementation of Data Logger.[Picture needed]

3.4.2.2 Route

Route is another basic structure how to define a dataflow from or to every module. Routes are defined with simple principle in mind, you define source and then every possible sink to which data have to be sent. By this configuration Data Logger will begin to accessing modules based on which side of the equation they are. If the module is generating data, framework will use specific general interface to gather data provided by the module and then distribute it to every sink defined by the route again by using general interface to sending data to module. With this principle one can define a very detailed data flow and by which assamble a data handling application in no time. If we would scale this framework to enourmous proportions, say we would have 10 sensors producing same data but they would need different modules to be handled, it would become soon tedious to write routes based only on id matching of modules. So for every module theres a specific setting to be set, a type, by which we can more easily define routes, e.g. `accelerometer -> database` and from now on, every data produced by module which has type set to acceloremeter will be routed to every module which as type of database, may there be big number of them.[Picture needed]

3.4.3 Data Logger modules

In this section of bachelor thesis I will desciribe modules which I developed in order to complete assignment.

3.4.3.1 Blank module

During developing modules for this bachelor thesis, I found myself using the same starting point for every module, so I created blank module which has all the neccesarry functions, or one might call it interface to make more easier creation of new modules, code is commented to give the reader neccesasry information what is expected from different function in case their name would not be clarifying enough. Blank module combines source and sink together but they do not need to be used both.

3.4.3.2 Time module

This is simple ‘dumb’ module which will be used to beta test others modules. Advantage of having this simiplistic module is it is not dependent on any hardware. Interval defined by sample rate will provide current time as data.

3.4.3.3 Accelerometer module

This module will be dependent on Adafruit 10 DOF sensor board and will utilize onboard LSM303 accelerometer sensor, communicating over I2C protocol. Module will emit proper acceleration in 3 axis, maximum acceleration +- 16g in each direction, sample rate will be configurable. LSM303 accelerometer sensor supports various settings as which axis acceleration is registered, how fast are these values checked, this constants for these settings are implemented, but I do not see the benefit of selecting only one axis or sampling at lower rate, so defaults are every axis is registered, and sampled as fast as it can be for accident checking module.

3.4.3.4 GPS module

GPS module as name suggests will parse nmea sentences from FONA 3g modem which has built-in GPS chip. AS for this module there is not much to be configured, because modem emits nmea sentences at 115200 rate, which to my best knowledge is not configurable, only thing found which could be configured is to switch output port for nmea sentences from nmea to serial, but then sending SMS and calling would not be possible, so it is not a feasible setting.

3.4.3.5 Accident module

This module will be both accepting input data and creating new information for other modules. New cars electronic crash sensors are accelometers. They are used to determine exactly when the airbag should be deployed to prevent injury to a driver or passenger. When an impact occurs, it results in vehicle deceleration, which is sensed by the accelerometer. This change is monitored by electronic control unit, which sends a signal to trigger the airbag. Not to rely on car accelerometer sensors the application will be polling LSM303 accelerometer sensor attached to the device. Problem with car accelerometer sensors is missing common interface to communicate with airbag/ECU system. If it is even possible then every car manafacturer has proprietary system in place, which would result in buying different specifications for every make, maybe even model. As for giving the gathered information value, I am not suitable for the task for simple reason, I am application comuter science undergrad and I can create, gather, store, act upon, etc based on information but to assign which value from accelerometer is right to register accident I have no knowledge, it is probably better task for someone from physics field. To create this module smart, in settings can be found treshold value which says that if accelerometer module registers acceleration over treshold accident module will be trigered and will compose a sos message to be sent to emergency dispatch centre. This message will be composed of latest GPS positon, maximal g-forces registred and medical information about driver. Medical information about driver, or in other words custom message is also configurable through settings.

3.4.3.6 SMS module

This module is a sms message sending module, but it is not designed for general use. Message relays on incoming data to be structured correctly from accident module and send 3 SMS messages with information about driver, GPS position and maximum g-force registered by the accelerometer. Modul requires modem to be pin unlocked otherwise SMS sending will fail. Sending SMS message with `AT+CGMS` command requires string to be sent be a maximal size of 150 characters and ended with `^Z` special character. After message is send, module repsonds with OK and number of total SMS messages sent by the module.

3.4.3.7 OBD-II module

By far the most challenging module in this thesis. Module connects with help of ELM327 compatible device to the OBD-II port and selects automatic protocol. Problem with automatic protocol selection is protocols have different initializing response, the module needed to be adapted to this discovery, to be working with older and newer cars, I successfuly tested this module on 5 different cars in make and model, but I can not rule out the possibility of OBD module not working with specific model and make of car, because there are multiple protocols present and their specification is not publicly available. After successful initialization, obd module will poll suported the car for supported commands, creates a polling loop with them and periodicaly queries data. One must be vary of this one limitation, and that is older cars have older protocols which have slow rate of sending data, and therefore sample rate of this module should be considered to be calculated correctly, otherwise buffer overflow might occurr. Module implements simple buffer overflow logic, when length of PID commands queue becomes more than treshold OBD module will not add more commands to queue until the length lowers than treshold. Module emits data for every queried command separately as in some cases gathering information from every supported command would simply took too long time to collect.

3.4.3.8 IFTTT module

IFTTT module is a support module for communicating with web service called Maker. With this service one can create different trigger events for data which arrives to the event and connect them to various other web services as GMail, Facebook, Instagram, etc. Data format is limited as Maker channel supports JSON format with only 3 keys, namely `value1,value2,vaule3`, so complicated data structures are not suited for this modul. Values of this 3 keys can be any string, with this in mind, the limitation could be circumvented for example with CSV format of data, which can be automatically parsed by form example google drive spreadsheet application. Fair to say is that this service is free of charge and provides with possibilities to connect data generated with vast number of popular web services.

3.4.3.9 Redis module

Redis module is database module for storing data in Redis database. As I mention above in Redis overview[TODO], redis stores data in mapping keys to values. For correct work module implements counters for every data source which wants to store data in database. Module stores whole given messages in hash keys. Key is composed from message header id and counter which is incremented by every message stored. As not to store counters in modules, counter current value is stored by message header id key. By configuring database snapshoting, redis database is able to recreate data structure from previous sessions, and continue where application left, with correct counters so data would not be accidentaly overriden. Database is password protected which is set by configuration file.

3.4.3.10 RPM module

RPM module is a module which listens for data from OBD-II module and evaluates only RPM(Revolutions per minute) data from car. Purpose of this module is to simulate modern cars which often provides gearshift indicator together with information when to shift. In configuration file can be find two values, downShiftTreshold is a treshold under which RPMs drops, module will generate data to indicate to shift gears, similiar action happens with upShiftTreshold, but instead dropping RPMs, it watchs when revolutions are higher then treshold. This module have also configured IFTTT event. This event triggers a notification on mobile device which has IFTTT application installed. In the screenshots below we can see what is looks like[TODO]

3.4.3.11 RabbitMQ module

RabbitMQ module is primary module for getting data from Rapsberry to server or cloud service. Module client connects to rabbitMQ server, after which it creates queues on the server based on message header id. Implication is that on server we will have queues for every module uploading data. Server then can process this queues separately with data processing applications and have various services to evaulate data and/or act upon it. In testing this module average speed of delivering messages was 500 messages in one second, this rate is based on how big information is being passed onto the server, so for example data from bulk module might have less throughput than accelerometer data. Connection settings can be altered in configuration file.

3.4.3.12 Console module

Very simple module which provides structured module for console logging data to show what information is passed into module. This module was primarily used for developing purposes same as time module. This module will probably not be used in production enviroment but is very handy for debuggind purposes, when modules do not cooperate in expected

way. Console header can be set only to show configuration capabilities in early stages of development.

3.4.3.13 Bulk module

Bulk module is module primarily used for grouping outputs from multiple modules into one big message. Reason for grouping messages is sending data over Internet rather in burst mode than constant stream. In bulk module settings user can choose time interval and size treshold for sending data. Whichever happens first triggers sending the message.

3.4.4 Data Logger routes

Data Logger routes are designed for interconnecting data streams from modules. These routes are designed so they would fullifil given assignment. Exact definiton how routes are set can be found in a configuration file. Main benefit of having these routes in one place and so easy configurable is that you can change complete outcome of the application by changing routes and redefinig them to suit your needs.

3.4.5 Main application

Main application is the core of ouput program, binds together every feature provided by the solution, and it is the 'brains' for every operation. Application is divisable by features which it offers. In upcoming sections I will describe every each one separatly to provide more detail.

3.4.5.1 Web Server

To provide web application for the user to be albe to configure Data Logger framework, server had to be created. This server works at port 8000 at localhost as to not confuse him with regular website. To connect to server, you have to be connected to the raspberry, either via LAN cable or to the wireless AP created automaticaly by the raspberry device. Password for this access point can be found in configuration file of hostapd at GitHub. Server supports websocets to communicate real-time data over them, also for configurating module settings.

3.4.5.2 Web Socket

For web application to display real time data produced by Data Logger modules websockets are used. For every module which produces data, main application will emit the information over websocket to every connected client. After connection client automatically starts listening for emitted info and display it accordingly. Through websockets it is possible to send new configuration to the server. Validating new configuration is burden for the web

application as websockets assumes that configuration is wirtten correctly.

3.4.5.3 Web Application

Once you are connected to the raspberry, open your favorite web browser and enter default ip address of raspberry device which if you did not change original configuration is `http://10.1.1.100:8000/`. After opening web application you will be presented with simple layout, at top navigation bar, you can switch between each module configuration, configuration of routes and main page. On main page you will be provided with real time data generated by the modules. In module pages current configuration is displayed which is queried by get request to the server. Server then reads current configuration from file, parsing it and sending it back to web applicaton. Module settings can be changed by clickng configure button. After this you will be presented by form which will allow to add,delete and configure settings. After finishing you can either confirm or cancel new settings. When confirmed, web application will check for syntax correctness. Not acting upon new configruation if it has errors or websocket will be used for sendning new configuration to the main application. After getting data to the server it's passed to the Data Logger framework to update changes to the running Data Logger and write them to the configuration file for changes to be persistent. For security reasons web application will be password protected as it can be found on costumer-grade routers.

4. Implementation

In this chapter of bachelor thesis I will provide a technical implementation details regarding my solution.

4.1 Overview

For completing this bachelor assignment I have choosen to implement it in Javascript programming language, which provides various benefits for its implementation.

- Over 70% of the code is implemented in JavaScript, rest is HTML and CSS.
- Application is operating system independent
- For backend solution I used Node.js, which is also implemented in JavaScript. So there is no need to know another programming language by the users
- The npm packgage registry holds over 250,000 reusable packages of reusable code.

As you can see there is a lot of support surrounding Node.js enviroment which enabled me to build better and faster application.

4.2 Data Logger

Data Logger is a standalone module. When Data Logger is required whole module is instantiated. For correct configuration it is needed to call configurate function with JavaScript Object as a argument which is passed by reading JSON configuration file from main application.

4.2.1 Modules

To process the file, modules have to be created by instantiating them with module options, which are then storted in internal key value object where keys are unique ids supplied by the

user. As this framework is primarily used for fellow computer scientists and power users, Data Logger does not implement any failsafe for not overwriting modules with same ID. This happend to be a mistake to forego this failsafe, and will be reported as one of many things to be improved in continued work after this thesis.

4.2.2 Routes

After modules are instantiated it is time for creating data flows defined by routes. To standardize data exchange format every message must be Javascript Object. Every message traveling between modules must be composed of these 2 key mapped values, firstly its header where the mandatory key is id, whichs value is reffering to the module in which message originated. Other properties of message header are optional and can be configured with main config file. Second mandatory key maped value is body, in which is stored actual data payload created by original module.[TODO TODO TODOTODOTODOOOO TODODO TABLE]
To better understand what interfaces must be implemented by the modules so they could create data or accept data I provide them in specification of interfaces below.

4.2.2.1 Interface of data creating module

For data created by the module to by properly handled by Data Logger module object must be a instance of the EventEmmiter class which exposes an eventEmitter.on() functions that allows for registering event listeners to that particular object. When object emits a specificly named event every function attached by on() function are called synchronously. So for data to be picked up be the Data Logger module must emit ‘data’ event witch argument is created data by the module. Nothing more is mandatory to by implemented.

4.2.2.2 Interface of data accepting module

Module which will be accepting data must be an instance of Stream Writable abstract class designed to be extended with an underlying implementation of the ._write method. This method must be implemented to accept data passing to the underlying resource in case of this thesis – module. Method _write has 3 arguments, (chunk, encoding, callback). Data intended for the module is in chunk variable.

5. Conclusion

Cieľom diplomovej práce bolo...

V práci som.....

Ďalší možný rozvoj....

Glossary

CAN Controller Area Network. 23, 24

ECU Electronic Control Unit. 22, 23

EPA Environmental Protection Agency. 24

I^{2}C Single-board computer. 15, 16

IoT Internet of Things. 13, 14

LAN Local Area Network. 23

OBD-II On-Board Diagnostics port. 12, 25, 26

OEM Original equipment manufacturer. 25

PIDs Parameter IDs. 26

SAE Society of Automotive Engineers. 24

SBC Single Board Computer. 16, 18–20

Bibliography

[1] N. Alee, M. Rahman, and R. B. Ahmad. “Performance comparison of Single Board Computer: A case study of kernel on ARM architecture”. In: *Computer Science Education (ICCSE), 2011 6th International Conference on*. Aug. 2011, pp. 521–524. DOI: 10.1109/ICCSE.2011.6028693.

[2] Tatiana De Almeida. *About Automatic*. URL: https://www.automatic.com/press/ (visited on 04/20/2016).

[3] Nagaraj.C Amit.V.Kachavimath. “Vehicle Accident Detection and Reporting System using Accelerometer”. In: *JIJSRD* 3.4 (2015). ISSN: 2321-0613.

[4] G. Audisio. “The Birth of the Cyber Tyre”. In: *IEEE Solid-State Circuits Magazine* 2.4 (Fall 2010), pp. 16–21. ISSN: 1943-0582. DOI: 10.1109/MSSC.2010.938647.

[5] Automatic. *How Automatic works*. URL: https://www.automatic.com/how-automatic-works/ (visited on 04/20/2016).

[6] Information is Beautiful. *Codebases, Milions of lines of code*. URL: http://www.informationisbeautiful.net/visualizations/million-lines-of-code/ (visited on 04/20/2016).

[7] Intel Corporation. *Edison Overview*. URL: http://download.intel.com/support/edison/sb/edison101presentationfromidfforthecommunity.pdf (visited on 04/20/2016).

[8] Dr. Peter Friess Dr. Ovidiu Vermesan. *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*. River Publishers, 2013. ISBN: 9788792982964.

[9] Albert Einstein. *Relativity: The Special and General Theory*. New York: Henry Holt, 1920. ISBN: 1587340925.

[10] M. Farsi, K. Ratcliff, and M. Barbosa. “An overview of controller area network”. In: *Computing Control Engineering Journal* 10.3 (June 1999), pp. 113–120. ISSN: 0956-3385. DOI: 10.1049/cce:19990304.

[11] Raspberry Pi Foundation. *What is a Raspberry Pi*. URL: https://www.raspberrypi.org/help/what-is-a-raspberry-pi/ (visited on 04/20/2016).

[12] http://i2c.info/. *I2C Bus Specification*. URL: http://i2c.info/i2c-bus-specification (visited on 04/20/2016).

[13] *I2C-bus specification and user manual*. Rev. 6. NXP Semiconductors. Apr. 2014.

[14] Adafruit Industries. *Adafruit 10-DOF - L3GD20H + LSM303 + BMP180*. URL: https://www.adafruit.com/products/1604 (visited on 04/20/2016).

[15] Adafruit Industries. *Adafruit 10-DOF - L3GD20H + LSM303 + BMP180*. URL: https://www.adafruit.com/products/2691 (visited on 04/20/2016).

[16] Yun-Sik Yu Mi-JinKim Jong-Wook Jang. "A Study on In-Vehicle Diagnosis System using OBD-II with Navigation". In: *International Journal of Computer Science and Network Security* 10.9 (Sept. 2010), pp. 136–140. ISSN: 1738-7906.

[17] Jason Kridner. *What is Beagle?* URL: http://beagleboard.org/brief (visited on 04/20/2016).

[18] G. Leen and D. Heffernan. "Expanding automotive electronic systems". In: *Computer* 35.1 (Jan. 2002), pp. 88–93. ISSN: 0018-9162. DOI: 10.1109/2.976923.

[19] G. Leen, D. Heffernan, and A. Dunne. "Digital networks in the automotive vehicle". In: *Computing Control Engineering Journal* 10.6 (Dec. 1999), pp. 257–266. ISSN: 0956-3385. DOI: 10.1049/cce:19990604.

[20] "News Briefs". In: *Computer* 47.7 (July 2014), pp. 16–21. ISSN: 0018-9162. DOI: 10.1109/MC.2014.189.

[21] Roger Penrose. *The Principle of Equivalence*. Knopf, 2014. ISBN: 0470085789.

[22] PTC. *Reducing Mean Time to Repair by 50 Percent with SmartConnect™*. 2015. URL: http://www.ptc.com/internet-of-things/customer-success/varian-medical-systems/thank-you (visited on 04/20/2016).

[23] J. Rui and S. Danpeng. "Architecture Design of the Internet of Things Based on Cloud Computing". In: *2015 Seventh International Conference on Measuring Technology and Mechatronics Automation*. June 2015, pp. 206–209. DOI: 10.1109/ICMTMA.2015.57.

[24] International Organization for Standardization. *Road vehicles – Diagnostic communication over Controller Area Network*. 2014. URL: http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=46045 (visited on 04/20/2016).

[25] Z. Szalay et al. “ICT in road vehicles #2014; Reliable vehicle sensor information from OBD versus CAN”. In: *Models and Technologies for Intelligent Transportation Systems (MT-ITS), 2015 International Conference on*. June 2015, pp. 469–476. DOI: 10.1109/MTITS.2015.7223296.

[26] *VI Monitor Features*. URL: http://www.vi-performance.com/?q=node/2 (visited on 04/20/2016).

Attachments

Automatic Hardware Specification

- OBD-II (conforms to J1962 standard), works with most cars since ’96
- Weight and Dimensions: 1.65 x 1.96 x 0.78in (42 x 50 x 20mm) 0.84oz (23.75g)
- Temperature Tolerance: Operating: -40 °F (-40 °C) to 158 °F (70 °C), Resting: -40 °F (-40 °C) to 185 °F (85 °C)
- Wireless: Bluetooth 4.0 Dual Mode (EDR and BLE), Made for iPhone (MFi) Certified
- Accelerometer: Frequency: 100Hz, Precision: 0.012G
- Security: Wireless encryption: 128-bit AES, Signed binary enforcement: 1024-bit RSA
- Built-in GPS: GNSS engine for GPS/QZSS, Logs trip routes with no phone present
- Audio Capabilities: Volume: 80db at 10cm at 2.5kHz
- Firmware: Over-the-air updates via smartphone

VI Monitor abilities

- Monitor parameters such as RPM, Speed, Throttle Position, Intake Manifold Pressure, Water Temperature, Air Fuel Ratios (lambda), Air Flow Rate, Ignition Advance Fuel Pressure, and many more.
- Perform braking and acceleration tests such as 0-60, 1/4 mile and 0-60-0 to measure your car’s true performance. Most tests can be G-triggered for unparrelled acuracy. Each test is recorded for future comparision.

- Avoid putting points on your license with adjustable speed warnings. Use the RPM warnings in conjunction with the adjustable Shift Light feature to get the most from your engines performance.
- Highly accurate G-Sensor with built-in damping monitors acceleration, brakiung and cornering G-Forces. Also records maximum G-readings.
- Record over 500 hours of engine and performance data on any parameters for review. Then upload the data to your computer for comparison. Ideal for measuring the effectiveness of modifications and recording drivers performance.
- Got a Engine Warning Light, but dont know why? - VI gives you the fault code number and a description of the problem, giving you more information to take to your garage or tuner.
- Simple stop/start timing feature allows you to record your times for later comparison on a computer.
- VI allows you to reset fault codes yourself, It also maintains a complete MIL stats history, including time and distance since the engine warning light was activated or rest.
- Using a Built-in Virtual Dynamomter, VI can test your vehicles true net horsepower in real world conditions.

Bluetooth / WiFi Combination USB Technical Details

- Ralink RT5370L WiFi adapter + BCM2046B1 Bluetooth 3.0 adapter
- Full-speed Bluetooth operation with Picone and Scatternet support
- Compliant with Bluetooth 2.1 + EDR and Bluetooth V3.0 + HS
- Support for WLAN + Bluetooth coexistence for optimized performance
- IEEE 802.11b/g/n compliance
- 150 Mbps Tx (transmit) PHY data rate
- 150 Mbps Rx (transmit) PHY data rate
- USB 1.1/2.0 compliant

npm packages

For clarification which packages I used during implementation, with short reasoning why I used it.

- **amqplib** – library used for making amqp 0.9.1 clients for Node.js, it has been used to create client for RabbitMQ module[3.4.3.11]
- **body-parser** – body parsing middleware, once internal part of express, now must be installed separately, middleware is used to handle POST requests
- **duplexer2** – library for creating writable and readable streams as one, used in intrenal structures of Data Logger.
- **express** – web application framework that provides a robust set of features for web and mobile applications.
- **express-session** – simple session middleware for Express, used in webserver middleware.
- **gps** – GPS.js is an extensible parser for NMEA sentences, used by GPS module to parse incoming data.[3.4.3.4]
- **i2c-bus** – I2C serial bus access used in accelerometer module[3.4.3.3]
- **lodash** – A modern JavaScript functional utility library delivering modularity, performance and extras. Library is used throughout whole application for correct handling with objects and arrays.
- **morgan** – HTTP request logger middleware for node.js, used to log http requests on the express server.
- **node-sass-middleware** – Connect middleware for node-sass, recompile .scss or .sass files automatically for connect and express based http servers.[citation needed]
- **object-sizeof** – library for computing size of a JavaScript object in bytes, used in bulk module to evaluate message size[3.4.3.13]
- **path** – this module contains utilities for handling and transforming file paths.[https://nodejs.org/docs/latest/api/path.html]
- **redis** – this is a complete and feature rich Redis client for node.js.[https://www.npmjs.com/package/redis] It is used in redis module[3.4.3.9]

- **request** – Request is designed to be the simplest way possible to make http calls. Used in IFTTT module to generate POST requests to the Maker channel[3.4.3.8]
- **serialport** – Node.js library to access serial ports, used in every module which required access to serial port.
- **socket.io** – node.js realtime framework server, used to communicate through websocekts or jsonp long polling
- **swig** – A simple, powerful, and extendable JavaScript Template Engine. Used in html templates for web application.[3.4.5.3][http://paularmstrong.github.io/swig/docs/]

Resources

CD obsahujúce:

- Elektronickú verziu
- Zdrojáky
- atď