|  |
| --- |
| UNIVERSIDADE REGIONAL DE BLUMENAU  CENTRO DE CIÊNCIAS EXATAS E NATURAIS  CURsO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO |
| OBD-JRP: MONITORAMENTO VEICULAR COM JAVA E RASPBERRY PI  RICARDO ARTUR STAROSKI |
| bLUMENAU  2016 |

|  |
| --- |
| RICARDO ARTUR STAROSKI  OBD-JRP: MONITORAMENTO VEICULAR COM JAVA E RASPBERRY PI  Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.  Prof. Miguel Alexandre Wisintainer - Orientador |
| bLUMENAU  2016 |
| OBD-JRP: MONITORAMENTO VEICULAR COM JAVA E RASPBERRY PI  Por  RICARDO ARTUR STAROSKI  Trabalho de Conclusão de Curso aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II pela banca examinadora formada por:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Presidente: Prof. Miguel Alexandre Wisintainer, Mestre – Orientador, FURB  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Membro: Prof(a). Nome do(a) Professor(a), Titulação – FURB  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Membro: Prof(a). Nome do(a) Professor(a), Titulação – FURB |
| Blumenau, dia de dezembro de 2016 |

Dedico este trabalho aos meus pais, pelo amor, apoio e compreensão por toda a vida.

AGRADECIMENTOS

Aos meus pais, pelo amor, apoio e compreensão, perante qualquer dificuldade, durante toda a vida.

À empresa Senior Sistemas, por flexibilizar meus horários de trabalho e disponibilizar sua infraestrutura para a realização deste trabalho.

Ao amigo e colega de equipe, Dennis Hiebert, por compreender e aceitar minhas ausências na empresa, para a realização deste trabalho.

Ao amigo e colega de equipe, Stephan Dieter Bieging, por ceder seu veículo para os testes de campo na realização deste trabalho.

Ao amigo Norberto Jensen, por ceder seu modem 3G para os testes de campo na realização deste trabalho.

Ao professor Miguel Alexandre Wisintainer, pela orientação, disponibilidade e entusiasmo em me ajudar e pelo e pelo auxilio extraclasse durante a realização deste trabalho.

Ao professor Maurício Capobianco Lopes, por me despertar o fascínio pelo desenvolvimento de jogos e pelo auxilio extraclasse durante a realização deste trabalho.

Ao professor Roberto Heinzle, pelo entusiasmo com que sempre ministrou as aulas de estruturas de dados e pelo auxilio extraclasse durante a realização deste trabalho.

Ao professor Aurélio Faustino Hoppe, pela paciência e apoio prestado durante meu reingresso ao curso.

Ao professor Mauro Marcelo Mattos, por ter me motivado com o processador virtual que desenvolvi para me ajudar no aprendizado de arquitetura de computadores.

À professora Joyce Martins, pela dedicação e entusiasmo em ajudar os alunos nas suas disciplinas e por me despertar o fascínio por compiladores.

Ao professor Alexander Roberto Valdameri, por tornar bastante interessante as aulas de bancos de dados, assunto com o qual não tenho afinidade.

Agradeço ainda a todas as pessoas que direta ou indiretamente contribuíram com meu crescimento acadêmico, profissional e pessoal.

Forja o teu espírito como o de uma espada, do mais forte aço e com o melhor fio, pois dele dependerá a sua vida.

Masaaki Hatsumi

RESUMO

O resumo é uma apresentação concisa dos pontos relevantes de um texto. Informa suficientemente ao leitor, para que este possa decidir sobre a conveniência da leitura do texto inteiro. Deve conter OBRIGATORIAMENTE o **OBJETIVO**, **METODOLOGIA**, **RESULTADOS** e **CONCLUSÕES**. O resumo deve conter de 150 a 500 palavras e deve ser composto de uma sequência corrente de frases concisas e não de uma enumeração de tópicos. O resumo deve ser escrito em um único texto corrido (sem parágrafos). Deve-se usar a terceira pessoa do singular e verbo na voz ativa (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2003).

Palavras-chave: Raspberry Pi, Internet das coisas, OBD2, IOT, Java, Bluetooth, Monitoramento veicular.

ABSTRACT

*Abstract* é o resumo traduzido para o inglês. *Abstract* vem em uma nova folha, logo após o resumo. Escrever com letra normal (sem itálico).

Key-words: Raspberry Pi, Internet of things, OBD2, IOT, Java, Bluetooth, Vehicular monitoring.

LISTA DE Figuras

[Figura 1 - Conector SAE J1962 e respectiva pinagem 13](#_Toc467796798)

[Figura 2 - Aspecto da interface ELM327 RS232 16](#_Toc467796799)

[Figura 3 - Aspecto da interface ELM327 USB 16](#_Toc467796800)

[Figura 4 - Aspecto da interface ELM327 Bluetooth 17](#_Toc467796801)

[Figura 5 - Aspecto da interface ELM327 WiFi 17](#_Toc467796802)

[Figura 6 - Blocos eletrônicos da interface ELM327 18](#_Toc467796803)

[Figura 7 - Visão geral dos protocolos de comunicação OBD 18](#_Toc467796804)

[Figura 8 - Características do Raspberry Pi 3 Model B 19](#_Toc467796805)

[Figura 9 - Conectando PyOBD com o veículo 21](#_Toc467796806)

[Figura 10 - Exibindo resultados de testes com PyOBD 21](#_Toc467796807)

[Figura 11 - Verificando dados em tempo real com PyOBD 22](#_Toc467796808)

[Figura 12 - Lendo e limpando códigos de falhas com PyOBD 22](#_Toc467796809)

[Figura 13 - Velocidade do veículo no EnviroCar 23](#_Toc467796810)

[Figura 14 - Velocidade média, trajeto e distância percorridos no EnviroCar 24](#_Toc467796811)

[Figura 15 - Informações coletadas pelo EnviroCar durante o percurso 24](#_Toc467796812)

[Figura 16 - Ciclo de vida do firmware 28](#_Toc467796813)

[Figura 17 - Ciclo de vida do servidor 29](#_Toc467796814)

[Figura 18 - Camadas e pacotes do firmware 30](#_Toc467796815)

[Figura 19 - Relacionamento entre as principais classes do firmware 33](#_Toc467796816)

[Figura 20 - Leitura de dados da interface ELM327 Bluetooth 35](#_Toc467796817)

[Figura 21 - Envio dos dados pendentes 36](#_Toc467796818)

[Figura 22 - Processamento de requisições no servidor 37](#_Toc467796819)

[Figura 23 - Relacionamento entre as classes do servidor 38](#_Toc467796820)

[Figura 24 - Versões do Sistema Operacional e Java no Raspberry Pi 39](#_Toc467796821)

[Figura 25 - Arquitetura da API JABWT BlueCove 40](#_Toc467796822)

[Figura 26 - Executando ObdJrpListDevices no Raspberry Pi 47](#_Toc467796823)

[Figura 27 - Executando ObdJrpScanData no Raspberry Pi 48](#_Toc467796824)

[Figura 28 - Interfaces ELM327 WiFi e Bluetooth 49](#_Toc467796825)

LISTA DE Quadros

[Quadro 1 - Comparativo entre os trabalhos correlatos e o trabalho proposto 25](#_Toc467796686)

[Quadro 2 - Listando dispositivos e serviços Bluetooth 42](#_Toc467796687)

[Quadro 3 - Disparando consulta de dispositivos com JABWT 43](#_Toc467796688)

[Quadro 4 - DiscoveryListener para consulta de dispositivos 43](#_Toc467796689)

[Quadro 5 - Classe Lock utilizada para sincronização de processos 44](#_Toc467796690)

[Quadro 6 - Disparando consulta de serviços com JABWT 45](#_Toc467796691)

[Quadro 7 - DiscoveryListener para consulta de serviços 46](#_Toc467796692)

[Quadro 8 - Arquivo obd-jrp.properties com configuração de acesso Bluetooth 48](#_Toc467796693)

[Quadro 9 - Método execute() da classe ObdJrpScanData 50](#_Toc467796694)

[Quadro 10 – Reestabelecendo conexão Bluetooth após erro 50](#_Toc467796695)

[Quadro 11 - Métodos restartAfterError() e stop() da classe ObdJrpScanData 51](#_Toc467796696)

[Quadro 12 - Atualizando interface de usuário em tempo real 51](#_Toc467796697)

[Quadro 13 - Apresentando dados lidos na interface de usuário 52](#_Toc467796698)

LISTA DE ABREVIATURAS E SIGLAS

API – Application Program Interface

CARB – Comitê de Administração dos Recursos do Ar da Califórnia

CONAMA – Conselho Nacional do Meio Ambiente

CSI – Camera Serial Interface

DSI – Display Serial Interface

DTC – Diagnostic Trouble Code

ECU – Electronic Control Unit

GPIO – General Purpose Input/Output

GUI – Graphic User Interface

HDMI – High Definition Multimedia Interface

IOT – Internet of Things

JABWT – Java API for Bluetooth Wireless Technology

JNI – Java Native Interface

JSR – Java Specification Request

LIM – Lâmpada Indicadora de Mau Funcionamento

M2M – Machine to Machine

MIL – Malfunction Indicator Lamp

OBD – On Board Diagnostic

PC – Personal Computer

PID – Parameter Identification Number

RAM – Random Access Memory

SAE – Society of Automotive Engineers

SD – SanDisk

SPP – Serial Port Profile

USB – Universal Serial Bus

UUID – Universally Unique Identifier

SUMÁRIO

[1 1 INTRODUÇÃO 9](#_Toc467796525)

[1.1 OBJETIVOS DO TRABALHO 10](#_Toc467796526)

[1.2 estrutura 10](#_Toc467796527)

[2 FUNDAMENTAÇÃO TEÓRICA 11](#_Toc467796528)

[2.1 HISTÓRIA DO OBD 11](#_Toc467796529)

[2.1.1 OBD NO BRASIL 11](#_Toc467796530)

[2.2 OBD2 12](#_Toc467796531)

[2.2.1 PROTOCOLOS OBD2 13](#_Toc467796532)

[2.2.2 MODOS DE DIAGNÓSTICO 14](#_Toc467796533)

[2.3 INTERFACE ELM327 15](#_Toc467796534)

[2.4 RASPBERRY PI 19](#_Toc467796535)

[2.5 TRABALHOS CORRELATOS 20](#_Toc467796536)

[2.5.1 PYOBD 20](#_Toc467796537)

[2.5.2 ENVIROCAR 22](#_Toc467796538)

[2.5.3 COMPARATIVO ENTRE OS TRABALHOS 25](#_Toc467796539)

[3 DESENVOLVIMENTO DO PROTÓTIPO 26](#_Toc467796540)

[3.1 requisitos 26](#_Toc467796541)

[3.2 ESPECIFICAÇÃO 27](#_Toc467796542)

[3.2.1 ESPECIFICAÇÃO DO FIRMWARE 29](#_Toc467796543)

[3.2.2 ESPECIFICAÇÃO DO SERVIDOR 36](#_Toc467796544)

[3.3 IMPLEMENTAÇÃO 38](#_Toc467796545)

[3.3.1 Técnicas e ferramentas utilizadas 38](#_Toc467796546)

[3.3.2 Operacionalidade da implementação 53](#_Toc467796547)

[3.4 ANÁLISE DOS RESULTADOS 53](#_Toc467796548)

[4 CONCLUSÕES 54](#_Toc467796549)

[4.1 EXTENSÕES 54](#_Toc467796550)

[Referências 55](#_Toc467796551)

[APÊNDICE A – Relação dos formatos das apresentações dos trabalhos 57](#_Toc467796552)

[ANEXO A – Representação gráfica de contagem de citações de autores por semestre nos trabalhos de conclusões realizados no Curso de Ciência da Computação 58](#_Toc467796553)

# 1 INTRODUÇÃO

A Internet das Coisas, ou Internet of Things (IOT), se refere a uma revolução tecnológica que tem como objetivo conectar os itens usados do dia a dia à rede mundial de computadores (ZAMBARDA, 2014). Cada vez mais surgem eletrodomésticos, meios de transporte e até mesmo roupas conectadas à internet e a outros dispositivos, como computadores e smartphones. Segundo GSM Association (2014), soluções Machine to Machine (M2M), já utilizam redes sem fio para conectar dispositivos uns aos outros e à internet, com o mínimo de intervenção humana. A IOT é uma evolução do M2M e representa a coordenação de máquinas, dispositivos e aparelhos de vários fornecedores conectados à internet através de múltiplas redes (GSM ASSOCIATION, 2014, tradução nossa).

Grande parte dos dispositivos domésticos incluem conectividade WiFi ou Bluetooth permitindo a comunicação com outros dispositivos e aparelhos (NG, 2015). Segundo Ng (2015), a capacidade de realizar análises em tempo real mudou para sempre a IOT, permitindo a implementação de sistemas preditivos e analíticos de forma eficiente. A principal aplicação dessas analises é auxiliar a identificar a causa raiz de falhas dos aparelhos, de forma a facilitar o processo de reparação (NG, 2015).

A especificação de um sistema capaz de recolher informações e estabelecer os diagnósticos de bordo é vantajosa para o dono do veículo, bem como para um técnico de reparação (ZURAWSKI, 2009, p. 33, tradução nossa). O termo utilizado para esta função é "diagnose de bordo" ou On Board Diagnostic (OBD). O conceito OBD refere-se ao auto diagnóstico do estado dos componentes do veículo. Segundo Zurawski (2009), o OBD só se tornou possível devido à introdução de sistemas computadorizados nos veículos. O papel das funções de diagnóstico predecessoras ao OBD era limitado a piscar uma luz assim que um problema específico fosse detectado. Zurawski (2009) explica que os sistemas OBD recentes são baseados na padronização da comunicação, dos dados monitorados e dos códigos de uma lista de falhas específicas.

CONAMA (2004) considera que o OBD, constitui tecnologia de ação comprovada na identificação de mau funcionamento de um veículo. Segundo CONAMA (2004), através da análise dos dados, é possível prevenir a ocorrência de avarias dos componentes do veículo.

Diante do exposto, este trabalho propõe o desenvolvimento de um protótipo de software embarcado em uma placa Raspberry Pi, para coletar informações da porta OBD de um veículo e disponibilizar estas informações em uma página web.

## OBJETIVOS DO TRABALHO

O objetivo deste trabalho é o desenvolvimento de um software embarcado, para coletar os dados da porta OBD2 de um carro, enviar estes dados para um servidor.

Os objetivos específicos do trabalho são:

1. desenvolver o firmware, que irá monitorar a porta OBD2 do carro, coletar dados e os enviar para um servidor;
2. desenvolver o software servidor, que irá receber os dados coletados pelo firmware e armazenar os mesmos;
3. desenvolver uma página web para consultar o histórico dos dados.

## estrutura

[Referir-se aos tópicos principais do texto, dando o roteiro ou ordem de exposição.]

# FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem como objetivo explorar os principais assuntos necessários para a realização deste trabalho. Os assuntos foram subdivididos em cinco partes, onde a seção 2.1 apresenta a origem do OBD. A seção 2.2 expõe detalhes técnicos do padrão OBD2. A seção 2.3 apresenta a interface ELM327. A seção 2.4 apresenta a plataforma Raspberry Pi e, por fim, na seção 2.5 são descritos dois trabalhos correlatos.

## HISTÓRIA DO OBD

On Board Diagnostic (OBD) significa Diagnose de Bordo. Este diagnóstico é realizado pelas próprias unidades eletrônicas do veículo. Segundo Manavella (2009), em 1988 o Comitê de Administração dos Recursos do Ar da Califórnia (CARB), estabeleceu uma norma não padronizada denominada OBD1 para que todos os veículos vendidos no estado da Califórnia, nos EUA, incorporassem em sua unidade de comando um sistema de diagnóstico capaz de detectar defeitos nos elementos e sistemas de controle de emissões. Manavella (2009) complementa que o OBD1 especificava um indicador luminoso chamado Malfunction Indicator Lamp (MIL), que acendia na presença de falhas. No Brasil o indicador MIL é chamado de Lâmpada Indicadora de Mau Funcionamento (LIM) (CONAMA, 2004).

### OBD NO BRASIL

No Brasil, o Conselho Nacional do Meio Ambiente (CONAMA), determinou a introdução dos sistemas de diagnose de bordo, em duas etapas complementares e consecutivas denominadas OBDBr-1 e OBDBr-2. De acordo com CONAMA (2004), o sistema OBDBr-1 foi implantado em sua totalidade em 1º de janeiro de 2009 e definiu as características mínimas para a detecção de falhas nos seguintes componentes, quando aplicável:

1. sensor de pressão absoluta ou fluxo de ar;
2. sensor de posição da borboleta;
3. sensor de temperatura de arrefecimento;
4. sensor de temperatura de ar;
5. sensor de oxigênio;
6. sensor de velocidade do veículo;
7. sensor de posição do eixo comando de válvulas;
8. sensor de posição do virabrequim;
9. sistemas de recirculação dos gases de escape;
10. sensor para detecção de detonação;
11. válvulas injetoras;
12. sistema de ignição;
13. módulo controle eletrônico do motor;
14. lâmpada indicadora de mau funcionamento; e
15. outros componentes que o fabricante julgue relevantes para a correta avaliação do funcionamento do veículo e controle de emissões de poluentes.

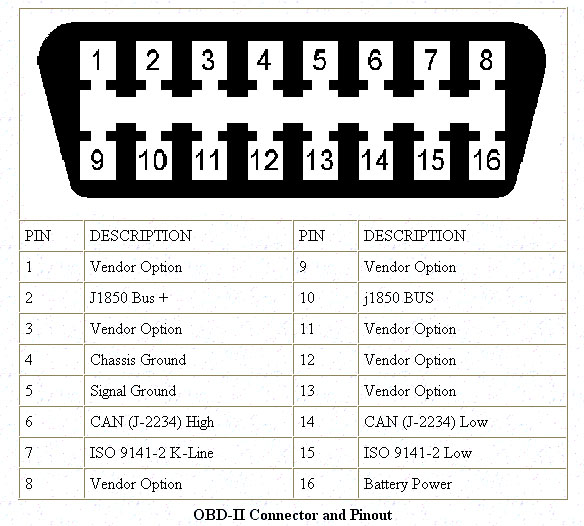
CONAMA (2004) considera que o sistema OBDBr-2 complementa as funções e características do sistema OBDBr-1. Segundo CONAMA (2004), o sistema OBDBr-2 deve detectar e registrar a existência de falhas, deterioração dos sensores de oxigênio e eficiência de conversão do catalisador. CONAMA (2004) complementa que o sistema OBDBr-2 deve apresentar características mínimas para a detecção de falhas nos seguintes componentes, quando aplicável:

1. sensores de oxigênio (pré e pós-catalisador);
2. eletroválvula do cânister; e
3. outros componentes que o fabricante julgue relevantes para a correta avaliação do funcionamento do veículo e controle de emissões de poluentes.

## OBD2

Não demorou muito para o CARB concluir que o padrão OBD1 não era eficiente para determinar o elemento que provocara o defeito. Portanto o CARB desenvolveu um novo conjunto de especificações, surgindo assim a norma OBD2 (MANAVELLA, 2009, p. 121). A Society of Automotive Engineers (SAE), estabeleceu a norma SAE J1962, que determinou o conector J1962 fêmea de 16 pinos, como a interface de hardware padrão para o OBD2. Na Figura 1 observa-se o aspecto e pinagem do conector J1962 (SAE INTERNATIONAL, 2006).

Figura 1 - Conector SAE J1962 e respectiva pinagem



Fonte: RioRand (2015).

Além do conector físico, a SAE também estabeleceu a norma SAE J1979, que define o método de requisição de dados de diagnóstico e uma lista dos parâmetros padrões disponíveis na Electronic Control Unit (ECU) (SAE INTERNATIONAL, 2006). Cada parâmetro é denominado Parameter Identification Number (PID) e cada código de erro é denominado Diagnostic Trouble Code (DTC). Conforme SAE International (2006), não é exigido que os fabricantes implementem todos os PIDs, é permitido a inclusão de PIDs proprietários, não listados na norma SAE J1979 e é permitido o acesso em tempo real aos PIDs e DTCs do veículo.

### PROTOCOLOS OBD2

Enquanto a porta OBD2 é normalizada em todo o mundo, vários protocolos de comunicação continuam possíveis, dependendo dos fabricantes de veículos (TOTAL CAR, 2014, tradução nossa). Atualmente estes protocolos podem ser classificados em três famílias: Redes CAN, Linhas K/L e SAE J1850.

#### REDES CAN

Segundo Total Car (2014), redes CAN utilizam os pinos 6 e 14 do conector J1962 e compreendem os seguintes protocolos:

1. ISO 157565: Utilizado por todos os veículos. Velocidade de comunicação de 125 a 500 Kbps;
2. SAE J1939: Utilizado principalmente por veículos pesados como caminhões e máquina agrícolas. Velocidade de comunicação de 125 a 500 Kbps.

#### LINHAS K/L

Segundo Total Car (2014), Linhas K/L utilizam os pinos 7 e 15 do conector J1962 e compreendem os seguintes protocolos:

1. ISO 9141-2: Utilizado principalmente por fabricantes europeus. Velocidade de comunicação de 10,4 Kbps;
2. ISO 14230 (KWP2000): Utilizado principalmente por fabricantes europeus. Dentro deste protocolo, existem dois sub protocolos que diferem no tempo de inicialização.

Slow init, “inicialização lenta” com velocidade de comunicação de 1,4 a 10,4 Kbps.

Fast init, “inicialização rápida” com velocidade fixa de 10,4 Kbps.

#### SAE J1850

Segundo Total Car (2014), SAE J1850 compreende os seguintes protocolos:

1. PWM: utilizado principalmente pela Ford Motors. Velocidade de comunicação de 41,6 Kbps. Utiliza os pinos 2 e 10 do conector J1962;
2. VPW: utilizado principalmente pela General Motors. Velocidade de comunicação de 10,4 a 41,6 Kbps. Utiliza somente o pino 2 do conector J1962.

### MODOS DE DIAGNÓSTICO

Independente do protocolo utilizado, o padrão OBD2 define 10 modos de diagnóstico. Não necessariamente todos os modos são suportados pelas ECUs. Quanto mais recente for o veículo, maior é a chance de haver suporte a mais modos (OUTILS OBD FACILE, 2015).

1. modo 1: Retorna valores comuns de alguns sensores como por exemplo, rotações do motor, velocidade do veículo, temperatura do motor, sensores de oxigênio e mistura ar/combustível. Cada sensor é identificado por um PID;
2. modo 2: Obtém o “instantâneo” de uma falha. Quando a ECU detecta uma falha, ela grava os dados do sensor daquele momento específico;
3. modo 3: Apresenta os DTCs armazenados. Segundo Outils OBD Facile (2015), estes códigos são padrão para todas as marcas de veículos e são divididos em quatro categorias:
   1. P0xxx: Para falhas associadas ao motor e transmissão,
   2. C0xxx: Para falhas associadas ao chassi,
   3. B0xxx: para falhas associadas à carroceria,
   4. U0xxx: para falhas associadas à comunicação de rede;
4. modo 4: Utilizado para apagar os DTCs gravados e desligar o MIL;
5. modo 5: Retorna o autodiagnostico do sensor lambda[[1]](#footnote-1). Segundo Outils OBD Facile (2015), este modo não é mais utilizado pois o modo 6 substitui suas funções;
6. modo 6: Retorna os resultados do autodiagnostico realizado nos diversos sensores do veículo;
7. modo 7: Este modo retorna DTCs não confirmados. Segundo Outils OBD Facile (2015), isto é bastante útil após um reparo no veículo, para confirmar que um DTC não está mais presente. Seus códigos são idênticos aos do modo 3;
8. modo 8: Segundo The Best OBD2 Scanners (2016), diferente dos outros modos que servem somente para ler informações, este modo é bidirecional, permitindo também gravar informações;
9. modo 9: Este modo obtém informações do veículo como por exemplo seu número de identificação;
10. modo 10: Este modo obtém os DTCs permanentes que, diferente dos modos 3 e 7, não podem ser apagados utilizando o modo 4. Outils OBD Facile (2015) explica que estes DTCs são apagados automaticamente pela própria ECU após rodar vários quilômetros sem que se repitam.

## INTERFACE ELM327

Segundo Total Car (2014), existem vários tipos de interface OBD2 e as mais comuns utilizam o circuito ELM327, de acordo com ELM Electronics (2016), o circuito ELM327 suporta todos os protocolos OBD2. Total Car (2014) explica que existem 4 tipos de interface ELM327:

1. ELM327 RS232: Conexão serial que está gradativamente desaparecendo nos computadores modernos. A Figura 2 apresenta o aspecto desta interface;
2. ELM327 USB: Conexão Universal Serial Bus (USB), presente na maioria dos computadores atuais. A Figura 3 apresenta o aspecto desta interface;
3. ELM327 Bluetooth: Conexão sem fio, que pode ser utilizada com computadores ou smartphones. A Figura 4 apresenta o aspecto desta interface;
4. ELM327 WiFi: Conexão sem fio que pode ser utilizada com computadores ou smartphones. Aspecto idêntico ao da interface Bluetooth, como ilustra a Figura 5.

Figura 2 - Aspecto da interface ELM327 RS232



Fonte: Total Car (2014).

Figura 3 - Aspecto da interface ELM327 USB



Fonte: Total Car (2014).

Figura 4 - Aspecto da interface ELM327 Bluetooth



Fonte: Total Car (2014).

Figura 5 - Aspecto da interface ELM327 WiFi

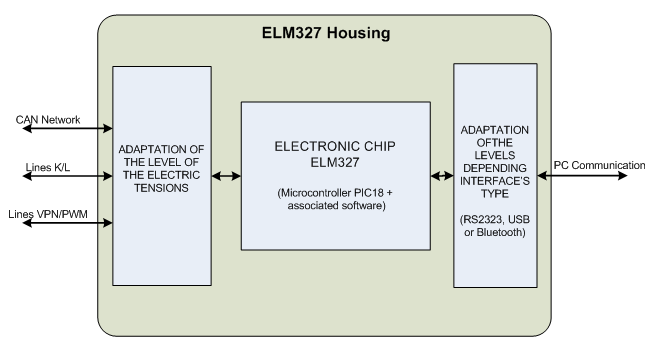


Fonte: Total Car (2014).

Apesar das aparências, estas 4 interfaces são eletronicamente idênticas. Somente o seu aspecto externo e o tipo de conexão são diferentes. No seu interior reside um circuito ELM327 (TOTAL CAR, 2014, tradução nossa). Para operar a interface, a unidade eletrônica é composta dos seguintes blocos, representados graficamente na Figura 6:

1. adaptadores de tensão elétrica: as redes on-board nos carros possuem níveis de tensão que requerem drivers específicos. Como o ELM327 suporta diversos protocolos, diversos drivers são necessários;
2. chip ELM327: é o circuito integrado, cujo nome é aplicado ao dispositivo como um todo. Ele seleciona o protocolo e o converte para um protocolo reconhecido por modems de computador. Ele atua como uma ponte entre os protocolos;
3. adaptadores de tensão para o computador: o chip por si só não é hábil para se comunicar com o computador, ele precisa adaptar os níveis de tensão antes de enviar o fluxo de dados.

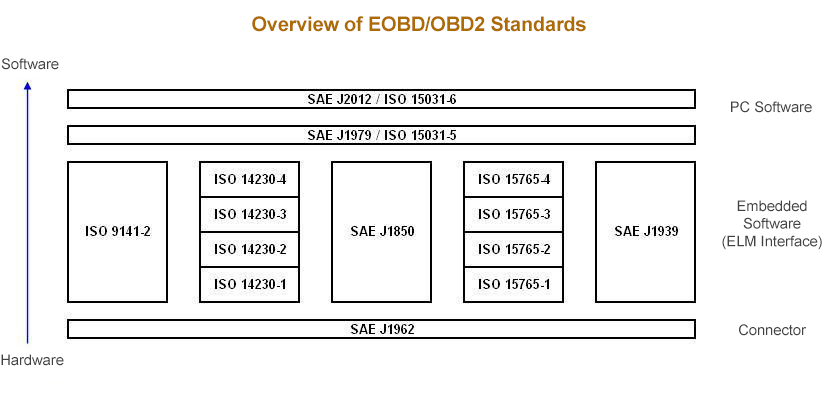
Figura 6 - Blocos eletrônicos da interface ELM327



Fonte: Total Car (2014).

Na Figura 7 observa-se a representação em colunas dos protocolos ISO 9141-2, ISO 14230, SAE J1850, ISO 15765 e SAE J1979. Segundo Total Car (2014), o papel do ELM327 é decodificar estes vários protocolos de comunicação.

Figura 7 - Visão geral dos protocolos de comunicação OBD

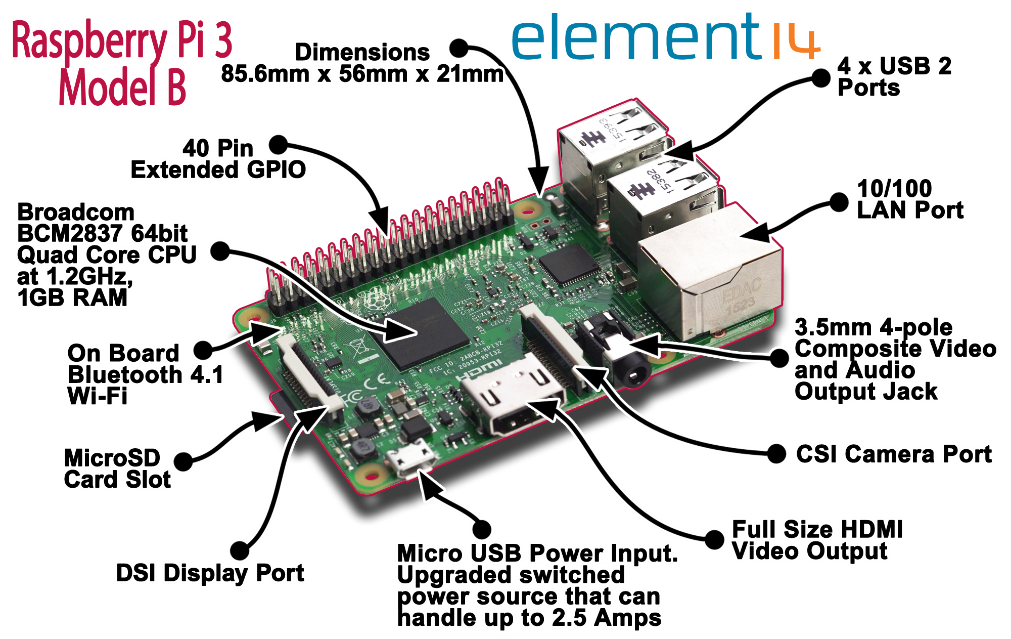


Fonte: Total Car (2014).

## RASPBERRY PI

O Raspberry Pi é um Personal Computer (PC), miniaturizado baseado no processador ARM. Ele pode realizar a maioria das tarefas que um desktop PC realiza, como por exemplo executar planilhas de cálculo, editores de texto e jogos (NEW IT LIMITED, 2016). Segundo Raspberry Pi Foundation (2016), ele foi desenvolvido para permitir que pessoas de todas as idades possam explorar a computação, aprender a programar e entender o funcionamento dos computadores. Na Figura 8 observa-se o aspecto da placa Raspberry Pi 3 Model B.

Figura 8 - Características do Raspberry Pi 3 Model B



Fonte: Thomsen (2016).

New IT Limited (2016) apresenta a seguinte especificação técnica da placa Raspberry Pi 3 Model B:

1. computador de placa única com chipset Broadcom BCM2837;
2. processador quad core ARM Cortex-A53 de 1,2GHz;
3. 1GB de Random Access Memory (RAM);
4. 40 pinos de General Purpose Input/Output (GPIO);
5. conexão Bluetooth 4.1 integrada;
6. conexão WiFi 802.11n integrada;
7. 1 porta Ethernet 10/100;
8. 4 portas USB;
9. 1 conector de 4 polos, combinado para saída de áudio estéreo e vídeo composto;
10. 1 saída High Definition Multimedia Interface (HDMI);
11. 1 porta Camera Serial Interface (CSI);
12. 1 porta Display Serial Interface (DSI);
13. 1 porta micro SanDisk (SD), para carga do sistema operacional e armazenamento de dados;
14. 1 porta micro USB para fonte de alimentação.

## TRABALHOS CORRELATOS

A seguir serão apresentados dois trabalhos correlatos ao trabalho proposto. O item 2.5.1 apresenta o PyOBD, uma ferramenta de diagnóstico automotivo compatível com OBD2 desenvolvida em linguagem de programação Python (PYOBD, 2015). O item 2.5.2 apresenta o EnviroCar, um aplicativo que permite compartilhar informações obtidas através da porta OBD2 (ENVIROCAR, 2015). O item 2.5.3 apresenta um quadro comparativo entre as características dos trabalhos correlatos e o trabalho proposto.

### PYOBD

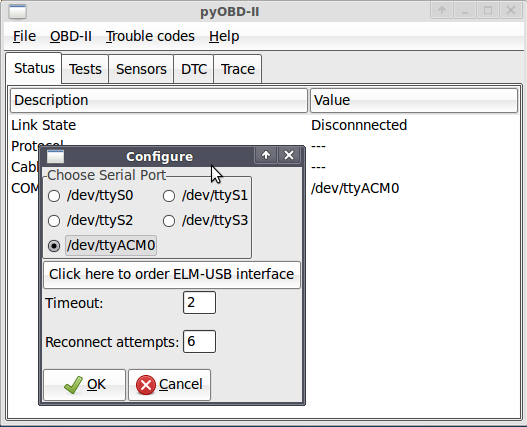
Trata-se de uma ferramenta open source de diagnóstico automotivo, segundo PyOBD (2015), a ferramenta foi projetada para se conectar à porta OBD2 através de uma interface ELM327 USB. PyOBD é voltado para desenvolvedores Python, é composto de um único módulo, chamado obd\_io, que permite um controle de alto nível sobre os dados dos sensores e gerenciamento dos códigos de erro (PYOBD, 2015). De acordo com PyOBD (2015), o módulo obd\_io foi testado para funcionar em notebooks ou desktop PCs com os sistemas operacionais Microsoft Windows, Linux e Mac OSX. Seus pré-requisitos são:

1. uma interface ELM327 USB;
2. python 2.x ou superior;
3. pacote py\_serial;
4. um veículo que implemente o padrão OBD2.

Com o PyOBD é possível:

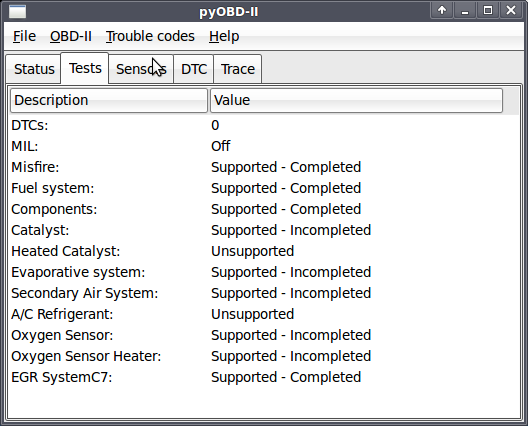
1. conectar-se ao veículo, conforme ilustrado na Figura 9;
2. exibir resultados de testes, conforme Figura 10;
3. verificar dados dos sensores em tempo real, conforme Figura 11;
4. ler e limpar códigos de falhas DTC, conforme Figura 12.

Figura 9 - Conectando PyOBD com o veículo



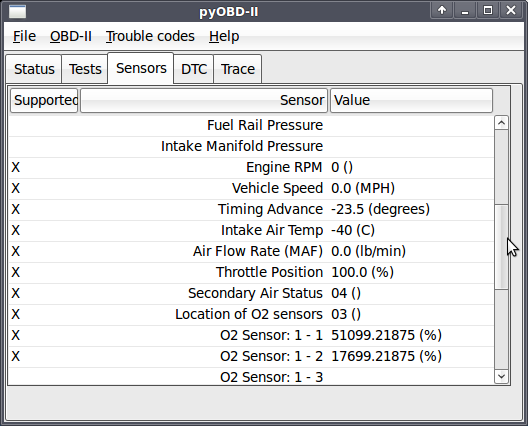
Fonte: PyOBD (2015).

Figura 10 - Exibindo resultados de testes com PyOBD



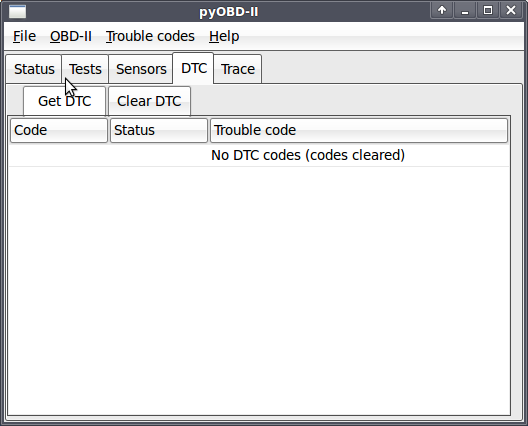
Fonte: PyOBD (2015).

Figura 11 - Verificando dados em tempo real com PyOBD



Fonte: PyOBD (2015).

Figura 12 - Lendo e limpando códigos de falhas com PyOBD



Fonte: PyOBD (2015).

### ENVIROCAR

Trata-se de um aplicativo alemão open source, desenvolvido para smartphones Android, seu propósito é que cidadãos, cientistas, engenheiros de tráfego e indústrias analisem dados OBD2 e compartilhem suas descobertas (ENVIROCAR, 2015, tradução nossa). O Aplicativo se conecta à porta OBD2 através de uma interface ELM327 Bluetooth. O usuário pode fazer upload das informações obtidas pelo aplicativo, diretamente para o servidor do EnviroCar. Segundo EnviroCar (2015), os dados ficam disponíveis anonimamente para que cientistas ou especialistas em tráfego acessem estes dados e os utilizem para solucionar questões ambientais e de mobilidade. EnviroCar permite que o usuário perceba o impacto ambiental causado pela forma de dirigir, investigando os dados dos sensores como consumo de combustível, emissão de gás carbónico e de ruídos (ANDROID PIT INTERNATIONAL, 2016).

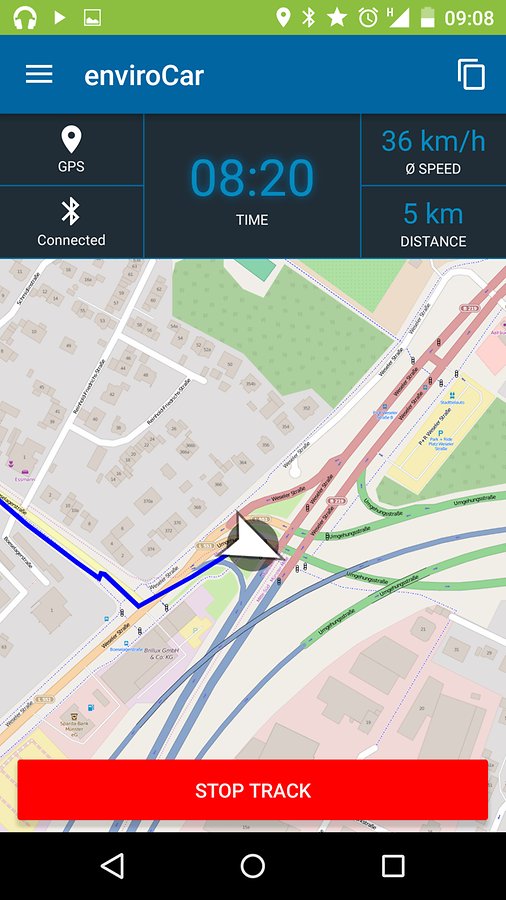
A seguir algumas telas do aplicativo EnviroCar. Na Figura 13 observa-se a velocidade do veículo em quilômetros por hora. A Figura 14 apresenta um mapa com o desenho do trajeto percorrido, o tempo da viagem, a distância percorrida e a velocidade média. Na Figura 15 são apresentadas diversas informações coletadas durante o percurso: marca e modelo do veículo, data e hora do início e término da viagem, consumo de combustível e a emissão de gás carbônico.

Figura 13 - Velocidade do veículo no EnviroCar



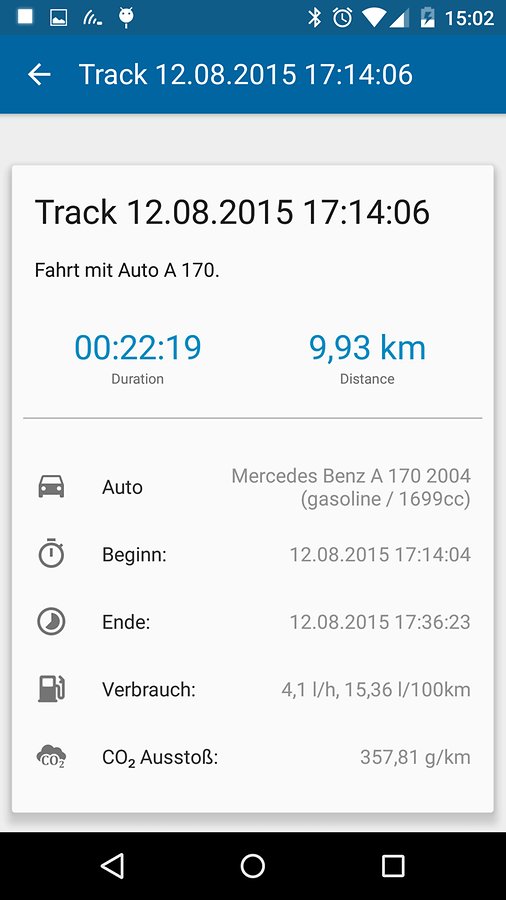
Fonte: Android Pit International (2016).

Figura 14 - Velocidade média, trajeto e distância percorridos no EnviroCar



Fonte: Android Pit International (2016).

Figura 15 - Informações coletadas pelo EnviroCar durante o percurso



Fonte: Android Pit International (2016).

### COMPARATIVO ENTRE OS TRABALHOS

O Quadro 1 apresenta um comparativo entre as características mais relevantes dos trabalhos correlatos apresentados e as características do trabalho proposto.

Quadro 1 - Comparativo entre os trabalhos correlatos e o trabalho proposto

|  |  |  |  |
| --- | --- | --- | --- |
| Características mais relevantes | Trabalhos Correlatos | | Trabalho Proposto |
| PyOBD | EnviroCar | OBD-JRP |
| Funciona em smartphone  (Sistema Android) |  | X |  |
| Funciona em notebook / desktop PC  (Sistemas Windows / Linux) | X |  | X |
| Funciona em dispositivo Raspberry Pi  (Sistema Raspbian) | X |  | X |
| Conecta-se à porta OBD2 através de interface ELM327 USB | X |  |  |
| Conecta-se à porta OBD2 através de interface ELM327 Bluetooth |  | X | X |
| Permite visualização dos dados OBD2 em tempo real | X | X | X |
| Faz upload dos dados coletados para um servidor externo |  | X | X |
| Disponibiliza histórico dos dados coletados em página web |  | X | X |
| Disponibiliza gráficos dos dados coletados em página web |  |  | X |

Observa-se no Quadro 1 que o trabalho proposto além de disponibilizar visualização dos dados coletados através de gráficos, procura combinar características distintas dos trabalhos correlatos, exceto o suporte à plataforma Android e a conexão por interface ELM327 USB.

# DESENVOLVIMENTO DO PROTÓTIPO

Neste capítulo são descritos os requisitos, a especificação do firmware e do servidor. Também é apresentada a implementação detalhando a operacionalidade do protótipo, os testes realizados em veículos reais. O capítulo finaliza com a descrição dos resultados obtidos.

## requisitos

Para simplificar a legibilidade, na descrição dos requisitos será utilizado o termo “firmware” para referenciar o software executando na placa Raspberry Pi instalada no veículo e o termo “servidor” para referenciar o software executando no servidor de aplicações TomCat. Os requisitos do protótipo a ser desenvolvido são:

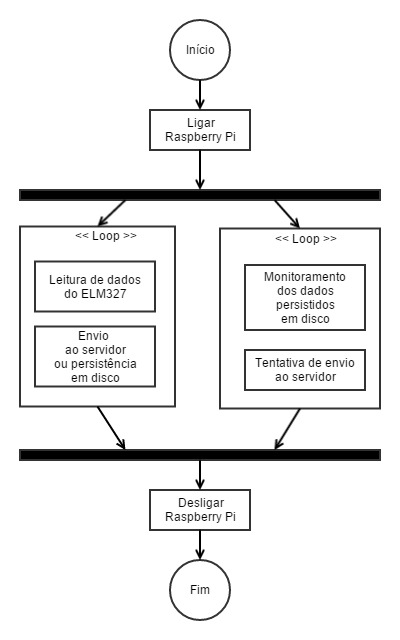
1. o firmware deverá ser inicializado automaticamente ao ligar a placa Raspberry Pi (Requisito Funcional – RF);
2. o firmware deverá se conectar à porta OBD2 através de uma interface ELM327 Bluetooth (RF);
3. o firmware deverá coletar os dados da porta OBD2 e armazená-los localmente até serem enviados ao servidor (RF);
4. o firmware deve tentar estabelecer uma conexão com o servidor a cada 5 minutos, caso não esteja conectado à internet (RF);
5. o firmware deverá enviar ao servidor o número do chassi do carro e os dados OBD2 armazenados localmente desde a última conexão bem-sucedida (RF);
6. o firmware deverá ser desenvolvido utilizando tecnologia Java SE (Requisito Não Funcional – RNF);
7. o firmware deverá executar em sistema operacional Raspbian (RNF);
8. o servidor deverá responder requisições HTTP, através dos métodos GET e POST[[2]](#footnote-2) (RF);
9. o servidor deverá persistir os dados coletados pelo firmware (RF);
10. o servidor deverá persistir os dados em arquivos XML, sem a necessidade de utilizar banco de dados (RNF);
11. o servidor deverá dispor uma página web para consultar os dados OBD2 a partir do número do chassi do carro (RF).
12. o servidor deverá ser desenvolvido utilizando tecnologia Java EE (RNF);
13. o servidor deverá executar no servidor de aplicações Apache TomCat (RNF);
14. a página web deve apresentar gráficos com os valores dos dados coletados (RF);
15. a página web deve apresentar uma tabela com os valores dos dados coletados (RF);
16. a página web deve ter interface responsiva de modo que possa ser visualizada em smartphones (RNF);
17. a página web deverá ser desenvolvida utilizando HTML, CSS e JavaScript (RNF).

## ESPECIFICAÇÃO

A solução consiste no desenvolvimento de um firmware embarcado em uma placa Raspberry Pi que se comunica com uma interface ELM327 Bluetooth para obter dados OBD2 e com um servidor para o qual estes dados são enviados. Inicialmente será apresentada a especificação do firmware e posteriormente a especificação do servidor.

O ciclo de vida do firmware consiste em disparar dois processos paralelos após a inicialização do Raspberry Pi, um processo é responsável pela leitura dos dados da interface ELM327 Bluetooth e o outro processo é responsável por monitorar o diretório onde os pacotes com os dados das leituras são persistidos. A Figura 16 apresenta o diagrama correspondente ao ciclo de vida do firmware.

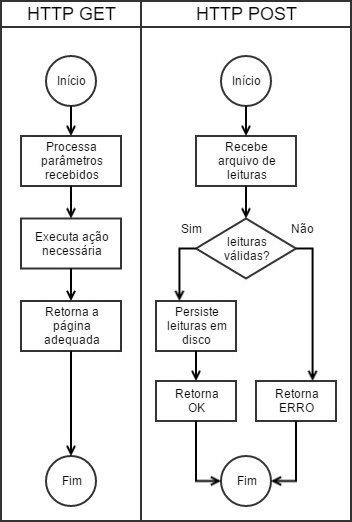
Figura 16 - Ciclo de vida do firmware



Fonte: Elaborado pelo autor.

O ciclo de vida do servidor consiste em processar requisições HTTP GET para retornar páginas solicitadas e processar requisições HTTP POST para receber os arquivos com leituras de dados enviadas pelo firmware. A Figura 17 apresenta o diagrama de atividades correspondente ao ciclo de vida do servidor.

Figura 17 - Ciclo de vida do servidor

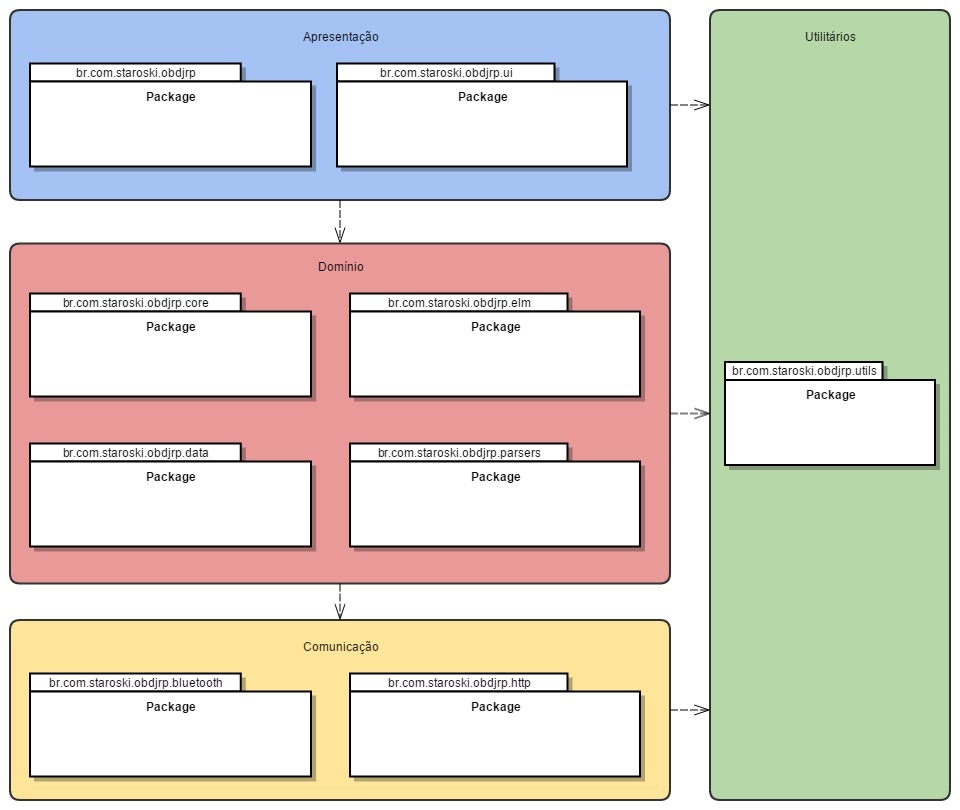


Fonte: Elaborado pelo autor.

### ESPECIFICAÇÃO DO FIRMWARE

O desenvolvimento do firmware foi dividido em 4 camadas distintas: apresentação, domínio, comunicação e utilitários. A camada de apresentação é responsável por fornecer o ponto de entrada para a execução dos processos no firmware e a interface de usuário, onde as leituras podem ser acompanhadas em tempo real. A camada domínio, como o nome sugere, contém as classes de domínio do protótipo. A camada de comunicação fornece classes que permitem a comunicação via Bluetooth, realização requisições HTTP GET e POST e na camada de utilitários residem classes de propósito geral, utilizadas pelas outras camadas. Na Figura 18 são apresentadas as camadas do firmware e a dependência entre eles, observa-se ainda quais os pacotes que compõe cada camada.

Figura 18 - Camadas e pacotes do firmware



Fonte: Elaborado pelo autor.

O pacote, br.com.staroski.obdjrp contém as classes que funcionam como ponto de entrada para a execução dos programas, na linguagem Java isso corresponde às classes que declaram um método com a assinatura public static void main(String[]). Neste pacote estão definidas três classes:

1. ObdJrpListDevices, programa que lista os dispositivos Bluetooth e seus serviços disponíveis;
2. ObdJrpScanData, programa que lê os dados da ECU através de uma interface ELM327 Bluetooth, tentando enviá-los ao servidor ou persistindo-os em disco;
3. ObdJrpUploadData, programa que monitora o diretório onde estão persistidas as leituras que o ObdJrpScanData não conseguiu enviar ao servidor e tenta reenviar estas leituras.

O pacote br.com.staroski.obdjrp.ui, contém a classe ScannerWindow, que representa uma Graphic User Interface (GUI), onde são apresentados em tempo real os dados lidos pelo programa ObdJrpScanData.

No pacote br.com.staroski.obdjrp.core estão as classes principais da API desenvolvida, são elas:

1. Config, classe que implementa o padrão de projeto Singleton[[3]](#footnote-3) e representa a configuração dos programas, que é realizada através de um arquivo texto chamado obd-jrp.properties;
2. DataMonitor, classe que monitora o diretório onde ficam os dados pendentes de envio e tenta enviá-las ao servidor, quem faz uso dessa classe é o programa ObdJrpUploadData;
3. IO, interface para objetos compostos de um InputStream para leitura de dados e um OutputStream para escrita de dados, é utilizada como parâmetro de construção para objetos do tipo Scanner;
4. Scanner, classe que realiza a leitura dos dados da ECU, comunicando-se com a interface ELM327, esta classe implementa o padrão de projeto Observer[[4]](#footnote-4), sendo possível registrar objetos para serem notificados quando as leituras são concluídas ou quando ocorrem erros;
5. ScannerListener, interface para os objetos que desejam receber notificações do Scanner.

As classes e interfaces supracitadas são somente as classes públicas declaradas no pacote, entretanto nele ainda estão declaradas outras seis classes não públicas: BluetoothIO, EventMulticaster, PackagePersister, ScanLoop, ScanUploader e SocketIO. Estas classes somente são utilizadas pelas classes públicas do pacote, de forma a aumentar a granularidade da implementação segregando as classes em partes menores de responsabilidade específica.

O pacote br.com.staroski.obdjrp.elm contém duas classes públicas, ELM327 e ELM327Error. A classe ELM327, como o nome sugere, representa uma interface ELM327, é através dela que são enviados os comandos ao harware conectado ao veículo. A classe ELM327Error representa um erro que pode ser lançado pela classe ELM327. Neste pacote existe ainda uma classe não pública chamada Disconnector, que se registra à máquina virtual Java através do método java.lang.Runtime.addShutdowHook() e quando a máquina virtual Java é encerrada, o Disconnector itera sobre a lista de objetos ELM327 ativos e os desconecta.

No pacote br.com.staroski.obdjrp.data encontram-se as classes responsáveis pela representação, serialização e persistência dos dados lidos através da interface ELM327. As classes são as seguintes:

1. Data, esta classe representa um dado lido da ECU, possui as propriedades pid e value, que contém o número do PID e os bytes lidos;
2. Scan, esta classe representa um objeto de leitura, que contém uma lista com os objetos Data lidos durante um ciclo de leitura;
3. Package, representa um pacote de dados pendentes de envio, cada Scan que não foi enviado com sucesso ao servidor, é adicionado a um Package;
4. Parser, é uma interface Strategy[[5]](#footnote-5) para objetos que implementam algoritmos que convertem objetos do tipo Data em um objetos do tipo Parsed;
5. Parsed, representa um objeto Data que foi processado por um Parser de forma a obter informação humanamente legível e possui as propriedades description e value, que contém a descrição e o valor;
6. Parsing, classe utilitária que utiliza implementações de Parser para transformar objetos Data em objetos Parsed.

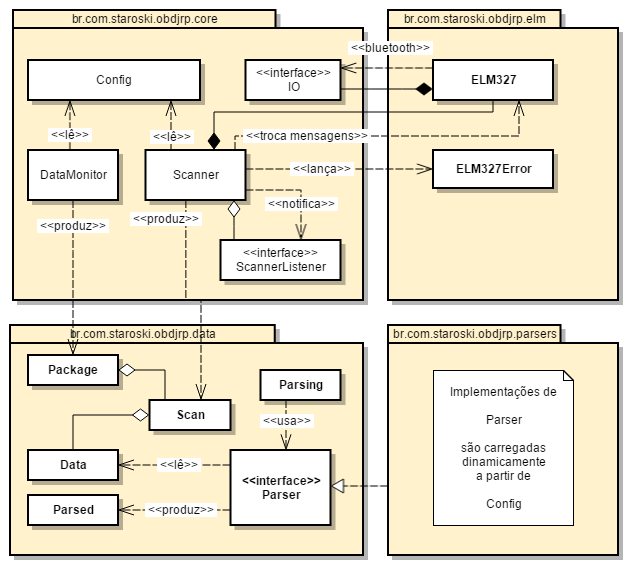
O pacote br.com.staroski.obdjrp.parsers define algumas classes que implementam a interface Parser. Como supracitado, um objeto Parser é responsável por converter um objeto Data em um objeto Parsed. Considerando o cenário hipotético de um objeto Data, com as propriedades pid=0C e value=0AF0, este objeto corresponde a uma leitura das Rotações por Minuto (RPM)[[6]](#footnote-6). A classe Parsing será utilizada para obter um objeto Parser adequado ao PID do objeto Data, vai executar o algoritmo do cálculo de RPM[[7]](#footnote-7), este objeto Parser vai gerar um objeto Parsed com as propriedades description=”Engine RPM” e value=2800, que é uma informação humanamente legível.

O pacote br.com.staroski.obdjrp.bluetooth define a classe Bluetooth, responsável por simplificar o acesso à API JABWT responsável pela descoberta de dispositivos Bluetooth e conexão aos mesmos. A classe Bluetooth implementa o padrão de projeto Façade[[8]](#footnote-8) e faz uso de uma classe não pública chamada DiscoveryAdapter, que provê uma forma simplificada de implementar a interface javax.bluetooth.DiscoveryListener.

No pacote br.com.staroski.obdjrp.http encontra-se a classe Http, que também é uma implementação do padrão de projeto Façade e provê uma interface simplificada para realizar requisições HTTP GET e HTTP POST. A classe Http realiza estas requisições respectivamente através de duas classes não públicas chamadas GetRequest e PostRequest.

Na Figura 19 é apresentado o digrama de pacotes do firmware e a forma como se relacionam as classes mais relevantes.

Figura 19 - Relacionamento entre as principais classes do firmware



Fonte: Elaborado pelo autor.

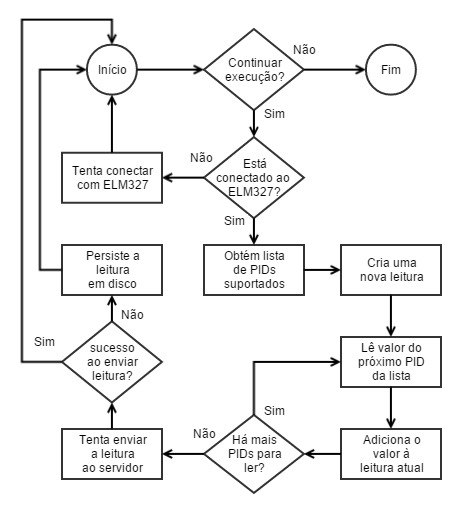
#### LEITURA DE DADOS DA INTERFACE ELM327 BLUETOOTH

Conforme citado na seção 3.2, o ciclo de vida do firmware consiste na execução de dois processos paralelos, o primeiro destes processos especificado, é a leitura dos dados da interface ELM327 Bluetooth. A leitura destes dados consiste em um laço que realiza as seguintes operações:

1. se estiver conectado ao ELM327, executa o passo c), senão executa o passo b);
2. tenta conectar-se ao ELM327 e volta ao passo a);
3. solicita ao ELM327 a lista dos PIDs suportados pelo veículo e executa o passo d);
4. cria um objeto de leitura para armazenar os valores dos PIDs e executa o passo d);
5. obtém o valor do próximo PID suportado e executa o passo f);
6. adiciona o valor do PID lido ao objeto de leitura e executa o passo g);
7. se houver mais PIDs para ler, volta ao passo e), senão executa o passo h);
8. tenta enviar o objeto de leitura ao servidor através de uma requisição HTTP GET e executa o passo i);
9. se conseguiu enviar o objeto de leitura, volta ao passo a), senão executa o passo j);
10. persiste o objeto de leitura em disco e volta ao passo a).

Na Figura 20 observa-se o fluxo do o processo de leitura da interface ELM327 Bluetooth.

Figura 20 - Leitura de dados da interface ELM327 Bluetooth



Fonte: Elaborado pelo autor.

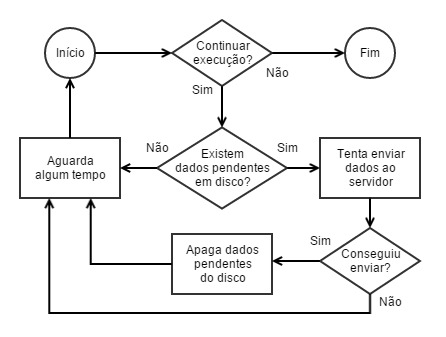
#### ENVIO DOS DADOS PENDENTES

O segundo processo do ciclo de vida do firmware é o envio dos dados pendentes, para este processo funcionar, é necessário que as configurações de rede do Raspberry Pi estejam possibilitando o acesso à internet. Não existe restrição quanto ao meio acesso, pode ser linha discada, DSL, 3G, WiFi, cabo ou outros. O processo de envio de dados trata de monitorar o diretório onde o processo de leitura dos dados persistiu as leituras que não puderam ser enviadas ao servidor. As leituras são empacotadas em um único arquivo e enviadas ao servidor através de uma requisição HTTP POST. Assim como a leitura dos dados da interface ELM327 Bluetooth, o envio dos dados pendentes também consiste em um laço que realiza as seguintes operações:

1. verifica se há arquivos de dados pendentes no diretório, se houver, executa o passo c), senão executa o passo b);
2. aguarda 5 minutos e volta ao passo a);
3. tenta enviar cada arquivo de dados pendentes ao servidor através de uma requisição HTTP POST e segue ao passo d);
4. apaga do disco, cada arquivo enviado com sucesso ao servidor, e volta ao passo b).

Na Figura 21 observa-se o fluxo do processo de transmissão dos dados pendentes.

Figura 21 - Envio dos dados pendentes

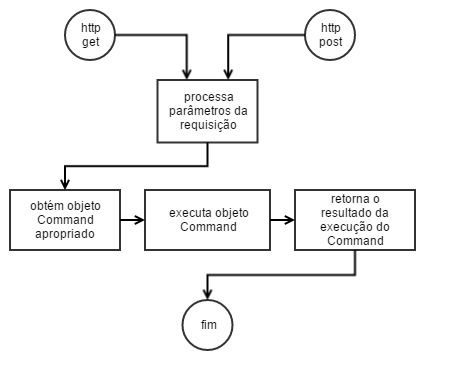


Fonte: Elaborado pelo autor.

### ESPECIFICAÇÃO DO SERVIDOR

O desenvolvimento do servidor foi realizado em uma única camada, que tem como ponto de entrada um Servlet Java EE, capaz de processar tanto requisições HTTP GET quanto requisições HTTP POST. Para processar as requisições, utilizou-se o padrão de projeto Command[[9]](#footnote-9), de forma que, a partir dos parâmetros recebidos, se obtenha um objeto apropriado para tratar a requisição. Na Figura 22 observa-se como ocorre o fluxo de processamento de requisições no servidor.

Figura 22 - Processamento de requisições no servidor



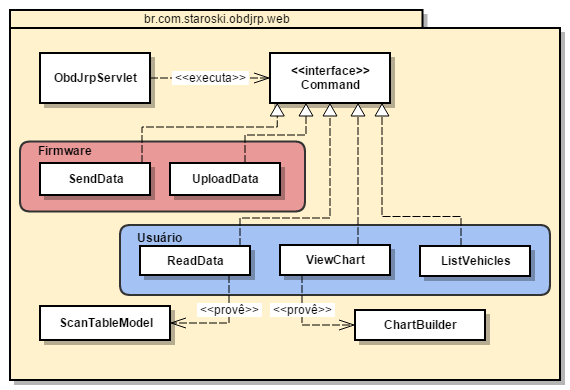
Fonte: Elaborado pelo autor.

O pacote que define as classes do servidor é chamado br.com.staroski.obdjrp.web, nele se encontram as seguintes classes:

1. ObdJrpServlet, o ponto de entrada do servidor, esta classe intercepta as requisições HTTP GET e HTTP POST, delegando a execução para uma implementação apropriada de Command;
2. Command, interface para os objetos que tratam parâmetros específicos das requisições recebidas pelo ObdJrpServlet;
3. SendData, implementação de Command invocada pelo firmware para enviar leituras em tempo real;
4. UploadData, implementação de Command invocada pelo firmware para enviar pacotes de leituras pendentes;
5. ListVehicles, implementação de Command invocada pelo usuário ao abrir a página com a lista de veículos;
6. ReadData, implementação de Command invocada pelo usuário ao abrir a página para visualizar leituras em tempo real;
7. ViewChart, implementação de Command invocada pelo usuário ao selecionar uma leitura para visualização gráfica;
8. ScanTableModel, provê métodos para a página renderizar uma tabela com as leituras;
9. ChartBuilder, provê métodos para a página renderizar os gráficos das leituras.

Na Figura 23 é apresentado o diagrama de classes do servidor. Observa-se que algumas classes estão delimitadas em uma região vermelha e outras em uma região azul, estas regiões respectivamente representam os recursos utilizados pelo firmware e os recursos utilizados pelo usuário ao acessar as páginas através de um navegador web.

Figura 23 - Relacionamento entre as classes do servidor



Fonte: Elaborado pelo autor.

## IMPLEMENTAÇÃO

Nesta seção são apresentados os aspectos sobre a preparação do ambiente de execução no Raspberry Pi, as implementações do firmware, do servidor, as ferramentas e técnicas utilizadas para a construção do protótipo.

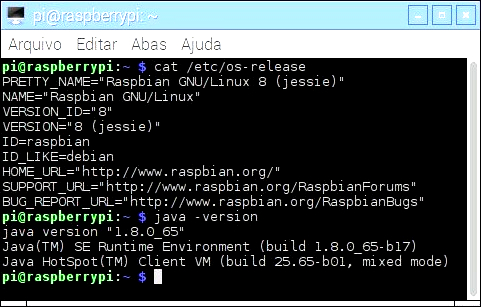
### Técnicas e ferramentas utilizadas

As implementações tanto do firmware quanto do servidor, foram realizadas utilizado o ambiente de desenvolvimento Eclipse Neon com linguagem de programação Java. Para o desenvolvimento do firmware foi utilizado a Application Program Interface (API) do Java SE e a API BlueCove para realizar a comunicação com a interface ELM327 Bluetooth. Para o desenvolvimento do servidor foi utilizado a API do Java SE, Java EE e API Google Charts, para criar gráficos em linguagem JavaScript. Os diagramas foram elaborados através da ferramenta Gliffy Online.

#### Preparação do ambiente de execução no Raspberry Pi 3 Model B

O sistema operacional instalado no Raspberry Pi foi o Raspian GNU/Linux 8, que é disponibilizada com a versão 1.8 do Java, como pode ser observado no terminal apresentado na Figura 24.

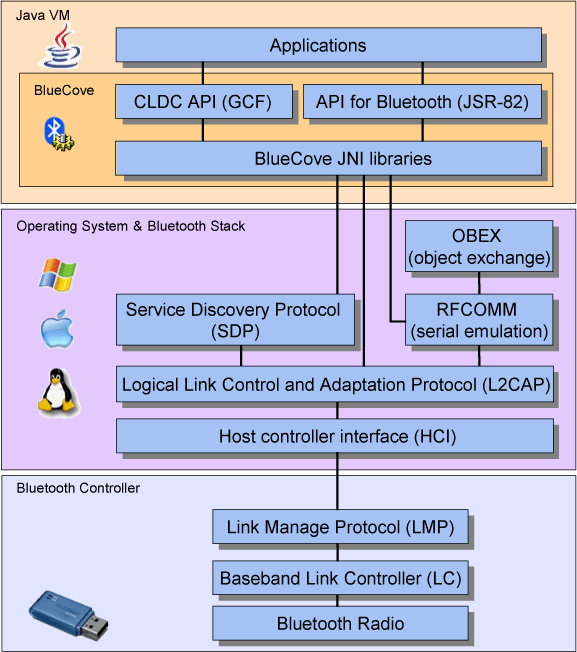
Figura 24 - Versões do Sistema Operacional e Java no Raspberry Pi



Fonte: Elaborado pelo autor.

Conforme citado na seção 2.4, o Raspberry Pi funciona de forma análoga à um PC e, para realizar a comunicação via Bluetooth com Java, foi utilizado a biblioteca BlueCove, que não faz parte da distribuição padrão do Java. BlueCove é uma Java API for Bluetooth Wireless Technology (JABWT), uma implementação da Java Specification Request 82 (JSR-82) (BLUECOVE, 2008, tradução nossa). A biblioteca BlueCove utiliza Java Native Interface (JNI) para a comunicação via Bluetooth, trocando mensagens diretamente com os drivers do sistema operacional, como pode ser observado na arquitetura apresentada na Figura 25.

Figura 25 - Arquitetura da API JABWT BlueCove



Fonte: BlueCove (2008).

Observou-se que no repositório[[10]](#footnote-10) da biblioteca BlueCove estão disponíveis versões compiladas para arquitetura x86, entretanto o Raspberry Pi possui arquitetura ARM, sendo necessário recompilar o código fonte no próprio dispositivo. Alderton (2015) explica que, para compilar o código fonte da biblioteca BlueCove no Raspberry Pi 2, é necessário instalar os pacotes bluetooth, bluez-utils e blueman. Este procedimento não funcionou no Raspberry Pi 3 Model B, somente o pacote bluetooth foi instalado com sucesso, já os pacotes bluez-utils e blueman não são suportados. Somente foi possível compilar o código fonte da biblioteca BlueCove após instalar os pacotes libbluetooth-dev, bluez, bluez-cups e bluez-obexd no Raspberry Pi. O processo de compilação do código fonte está detalhado nos arquivos read-me.txt e developer-read-me.txt, disponíveis no repositório da biblioteca BlueCove. Com base nas instruções destes arquivos, foi necessário:

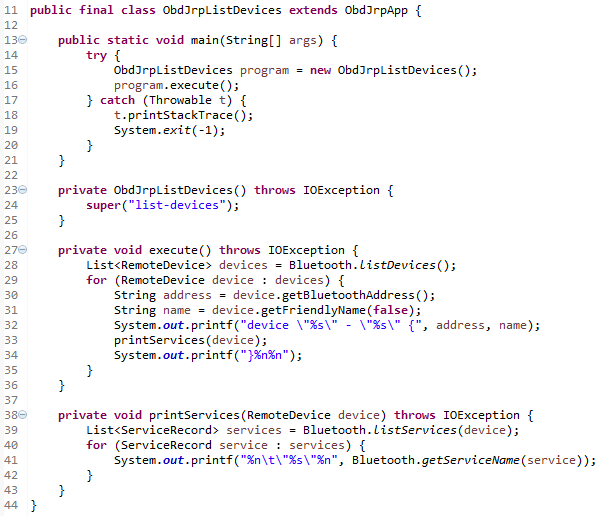
1. instalar as ferramentas Maven e Ant no Raspberry Pi;
2. executar o comando Maven para criar os diretórios de código fonte compatíveis com o ambiente Eclipse:
   1. mvn eclipse:clean eclipse:eclipse -DdownloadSources=true;
3. executar o comando Ant para compilar as bibliotecas nativas e gerar o arquivo JAR da biblioteca Java:
   1. ant all.

Com a biblioteca BlueCove compilada para arquitetura ARM, foi possível realizar no próprio Raspberry algumas provas de conceito para avaliar se seria viável dar continuidade ao desenvolvimento utilizando a linguagem Java. Os testes consistiram em listar os dispositivos Bluetooth pareados e tentar obter a lista de serviços Bluetooth disponíveis, como os testes foram positivos, deu-se continuidade ao desenvolvimento do protótipo utilizando a linguagem Java, caso os testes com o BlueCove não fossem positivos, uma alternativa seria pesquisar bibliotecas para comunicação Bluetooth da linguagem Python[[11]](#footnote-11), que também é disponibilizada com o Raspian GNU/Linux 8.

#### Listando dispositivos Bluetooth no firmware

A primeira aplicação desenvolvida para o firmware foi um programa em linha de comando, chamado ObdJrpListDevices, que lista os dispositivos Bluetooth pareados e os serviços Bluetooth disponíveis, este programa serviu como prova de conceito para a viabilidade de utilizar a linguagem Java para comunicação Bluetooth no Raspberry Pi. No Quadro 2 é apresentado o código fonte da classe ObdJrpListDevices, que procura obter a lista de dispositivos Bluetooth pareados e identificar os serviços que os mesmos disponibilizam.

Quadro 2 - Listando dispositivos e serviços Bluetooth

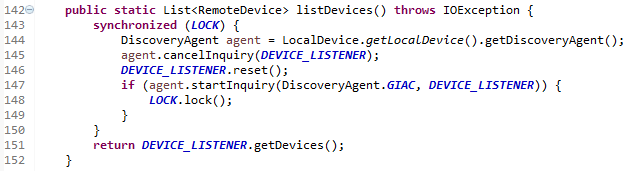


Fonte: Elaborado pelo autor.

Observa-se na linha 28 da classe ObdJrpListDevices, a invocação do método estático Bluetooth.listDevices(), A classe Bluetooth foi desenvolvida para abstrair a complexidade da JABWT implementada pela biblioteca BlueCove. Thompson, Kline e Kumar (2008, p. 136), explicam que antes de consultar um dispositivo, é necessário que a aplicação implemente a interface DiscoveryListener e os métodos deviceDiscovered() e inquiryCompleted().

Para realizar a consulta de dispositivos, é necessário invocar o método startInquiry() da classe DiscoveryAgent, passando por parâmetro a instância do DiscoveryListener. No Quadro 3 é apresentado o código fonte do método Bluetooth.listDevices(). Na linha 146 é invocado o método startInquiry(), passando como parâmetro o objeto DEVICE\_LISTENER, este objeto é uma instância de DiscoveryListener.

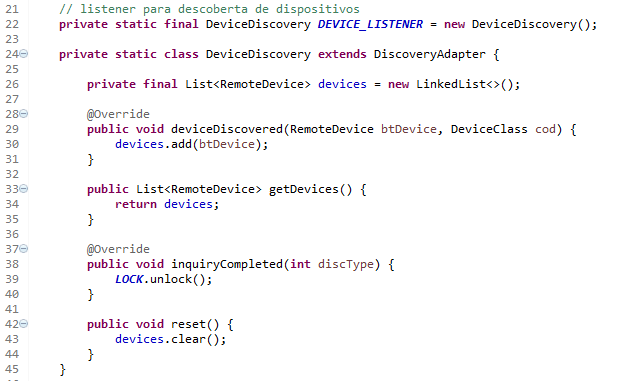
Quadro 3 - Disparando consulta de dispositivos com JABWT



Fonte: Elaborado pelo autor.

Observa-se na linha 147 que, caso a invocação do startInquiry() retorne true, é invocado o método LOCK.lock(), este método faz com que a execução do método Bluetooth.listDevices() permaneça bloqueada até que outra Thread invoque o método LOCK.unlock(). Esse bloqueio é necessário pois o método startInquiry() retorna imediatamente após ser invocado, esse retorno imediato ocorre pois a consulta de dispositivos ocorre concorrentemente em outra Thread disparada pelo startInquiry(), isso justifica a necessidade de registrar-se uma instância de DiscoveryListener para ser notificada pelo método deviceDiscovered() quando um dispositivo é encontrado e pelo método inquiryCompleted() quando a consulta termina. A implementação do DiscoveryListener utilizado é apresentada no Quadro 4.

Quadro 4 - DiscoveryListener para consulta de dispositivos

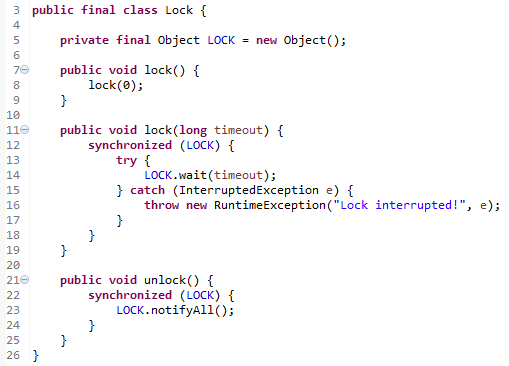


Fonte: Elaborado pelo autor.

Na linha 30 do DiscoveryListener, o dispositivo recebido pelo parâmetro btDevice é adicionado à lista devices e na linha 39, quando a consulta termina, é invocado o método LOCK.unlock(), de forma a desbloquear o método Bluetooth.listDevices() e retornar a lista de dispositivos descobertos.

Optou-se em escrever uma classe alternativa para sincronização de processos, pois os métodos java.lang.Object.wait(), java.util.concurrent.locks.Lock.lock() e java.util.concurrent.Semaphore.acquire(), declaram o lançamento da exceção checada InterruptedException, forçando o desenvolvedor a tratar ou relançar a exceção. Os métodos da classe Lock criada, não declaram o lançamento de nenhuma exceção checada, e o método lock() trata a InterruptedException e a transforma em uma exceção não checada do tipo RuntimeException, essa técnica torna mais limpo o código onde o método lock() for utilizado. No Quadro 5 observa-se a implementação da classe Lock.

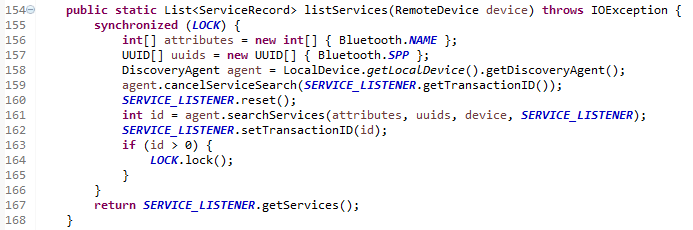
Quadro 5 - Classe Lock utilizada para sincronização de processos



Fonte: Elaborado pelo autor.

O processo de consulta de serviços utilizando JABWT é idêntico ao processo de lista consulta de dispositivos, é necessário invocar o método searchServices(), mas além de passar por parâmetro um objeto do tipo DiscoveryListener, também é necessário passar o objeto RemoteDevice do qual se deseja obter os serviços disponíveis, um array com os atributos que se deseja obter do serviços e um array de Universally Unique Identifier (UUID) correspondente ao perfil de serviço Bluetooth que se deseja listar. No trabalho proposto o único atributo de interesse é o nome do serviço e o perfil Serial Port Profile (SPP), segundo Bluetooth (2014, p. 1949), o valor hexadecimal correspondente ao atributo ServiceName é 0x0100 e segundo a documentação[[12]](#footnote-12) da classe javax.bluetooth.UUID, o valor hexadecimal correspondente ao perfil SPP é 0x1101. No Quadro 6 é apresentado a implementação do método Bluetooth.listServices(), que realiza a consulta dos serviços disponíveis para o dispositivo informado no parâmetro device.

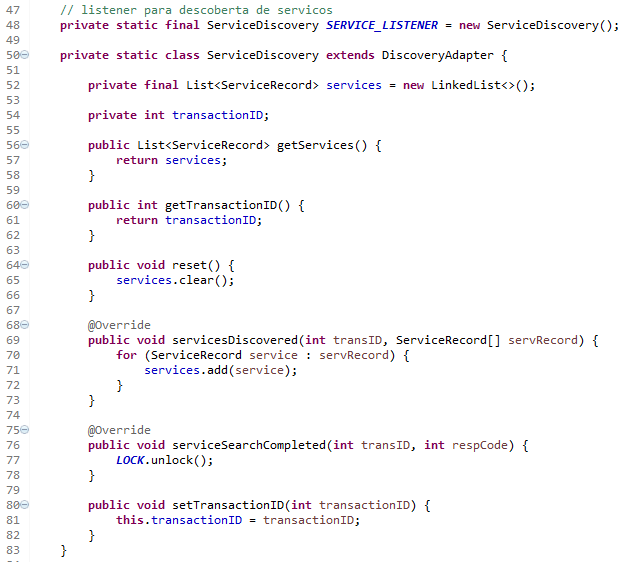
Quadro 6 - Disparando consulta de serviços com JABWT



Fonte: Elaborado pelo autor.

Observa-se entre as linhas 161 e 163 que caso a invocação do searchServices(), retorne um id maior que zero, é invocado o método LOCK.lock(), fazendo com que a execução do método Bluetooth.listServices () permaneça bloqueada até que outra Thread invoque o método LOCK.unlock(). Esse bloqueio é necessário pois o método searchServices() também é assíncrono e retorna imediatamente após ser invocado, também se faz necessário registrar uma instância de DiscoveryListener para ser notificada pelo método servicesDiscovered() quando serviços são descobertos e pelo método serviceSearchCompleted() quando a consulta termina. A implementação do DiscoveryListener utilizado para a descoberta de serviços é apresentada no Quadro 7.

Quadro 7 - DiscoveryListener para consulta de serviços

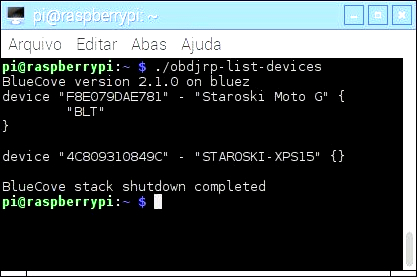


Fonte: Elaborado pelo autor.

Na linha 71 do DiscoveryListener, cada serviço descoberto recebido pelo parâmetro servRecord é adicionado à lista services e na linha 77, quando a consulta termina, é invocado o método LOCK.unlock(), de forma a desbloquear o método Bluetooth.listServices() e retornar a lista de serviços descobertos para o dispositivo informado.

Na Figura 25 observa-se o resultado da execução da classe ObdJrpListDevices no Raspberry Pi, listando dois dispositivos. O primeiro dispositivo possui o endereço F8E079DAE781, nome “Staroski Moto G” e disponibiliza um serviço chamado “BLT”. O segundo dispositivo possui endereço 4C809310849C, nome “STAROSKI-XPS15” e não possui nenhum serviço disponível.

Figura 26 - Executando ObdJrpListDevices no Raspberry Pi

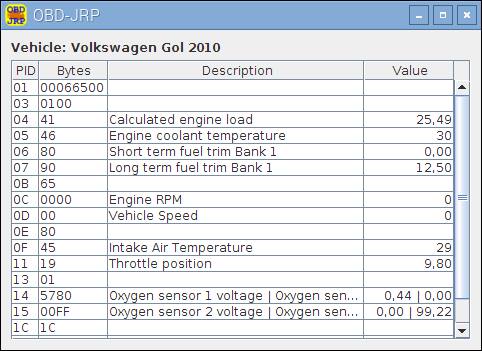


Fonte: Elaborado pelo autor.

#### Leitura de dados em tempo real

A segunda aplicação desenvolvida para o firmware foi um programa em linha de comando chamado ObdJrpScanData, que se comunica via Bluetooth com a interface ELM327 conectada à porta OBD2 do veículo. Embora o ObdJrpScanData seja um programa em linha de comando, ele apresenta uma interface gráfica contendo uma lista com os PIDs e respectivos valores lidos em tempo real. Esta janela foi desenvolvida pois tentou-se conectar um display touchscreen de 3,5 polegadas ao Raspberry Pi, de forma que as leituras dos PIDs pudessem ser observadas dentro do próprio veículo. Entretanto o display não apresentou compatibilidade com o Raspberry Pi 3. Mesmo assim, optou-se em manter a janela no programa de forma a auxiliar nos testes realizados, para visualizar a área de trabalho do Raspberry Pi utilizou-se um monitor com entrada HDMI e para acessar sua área de trabalho enquanto conectado ao veículo, utilizou-se o acesso removo através do VNC[[13]](#footnote-13). Na Figura 27 observa-se o aspecto da janela apresentada pelo ObdJrpScanData.

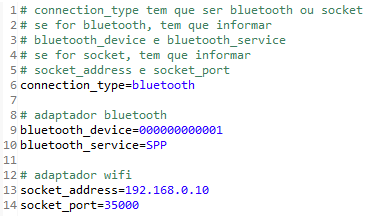
Figura 27 - Executando ObdJrpScanData no Raspberry Pi



Fonte: Elaborado pelo autor.

Para se comunicar com a interface ELM327 Bluetooth, o ObdJrpScanData precisa conhecer o endereço Bluetooth do ELM327 e o nome do serviço disponibilizado. Estas informações são configuradas em um arquivo chamado obd-jrp.properties que é representado pela classe Config. Para obter o endereço Bluetooth do dispositivo e o nome do serviço disponibilizado. Para descobrir o endereço do dispositivo e o nome do serviço, executou-se a classe ObdJrpListDevices enquanto a interface ELM327 estava conectada à um veículo. Com as informações obtidas, editou-se o arquivo obd-jrp.properties com as propriedades apresentadas no Quadro 8.

Quadro 8 - Arquivo obd-jrp.properties com configuração de acesso Bluetooth



Fonte: Elaborado pelo autor.

Observa-se que a primeira propriedade definida no arquivo obd-jrp.properties se é connection\_type, esta propriedade determina qual o tipo de conexão a ser utilizada. A implementação da classe Config só interpreta dois valores para o connection\_type: bluetooth e socket. O suporte para conexões via Socket foi implementado pois, enquanto não foi possível compilar a biblioteca BlueCove no Raspberry Pi, utilizou-se uma interface ELM327 WiFi. Entretanto tal abordagem se demonstrou inviável pois quando as configurações de rede do Raspberry Pi são modificadas para acessar o ponto de rede WiFi da interface ELM327, o dispositivo fica impossibilitado de conectar-se à internet. Na Figura 28 é apresentado o aspecto das duas interfaces utilizadas durante o desenvolvimento do protótipo. Percebe-se que, embora sejam acessíveis por meios diferentes, seu encapsulamento é semelhante.

Figura 28 - Interfaces ELM327 WiFi e Bluetooth



Fonte: Elaborado pelo autor.

Com as configurações de acesso definidas no arquivo obd-jrp.properties, a classe ObdJrpScanData obtém, através da classe Config, uma instância da interface IO. Independentemente do meio de acesso ser Bluetooth ou Socket, a classe Config sempre disponibiliza um objeto do tipo IO que abstrai o meio de comunicação. O objeto IO é parâmetro de construção para objetos do tipo Scanner. Após instancias um Scanner, pode-se registrar objetos do tipo ScannerListener, que serão notificados quando um ciclo de leitura for concluído ou quando ocorre um erro na comunicação com a interface ELM327. Para finalmente inicializar a leitura, é invocado o método start() do objeto Scanner. Este procedimento é executado pelo programa ObdJrpScanData na implementação do método execute(), como pode ser visto no Quadro 9.

Quadro 9 - Método execute() da classe ObdJrpScanData

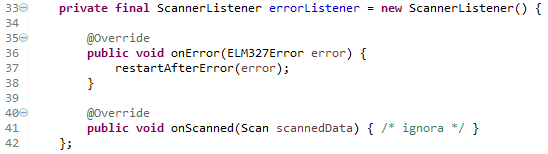


Fonte: Elaborado pelo autor.

Observa-se nas linhas 56 e 57 que a classe ObdJrpScanData registra dois objetos que implementam a interface ScannerListener:

1. errorListener, responsável por tentar reestabelecer a conexão com a interface ELM327 caso ocorra um erro, sua implementação é apresentada no Quadro 10;
2. windowListener, responsável por atualizar a janela que apresenta as leituras em tempo real, sua implementação é apresentada no Quadro 12.

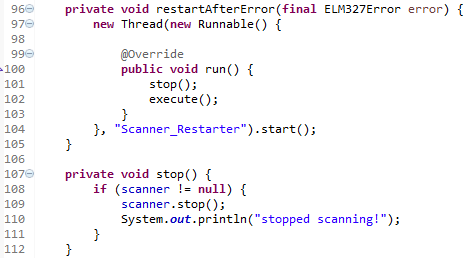
Quadro 10 – Reestabelecendo conexão Bluetooth após erro



Fonte: Elaborado pelo autor.

Observa-se que na linha 41 é ignorado o evento de leitura recebida, o ScannerListener apenas trata o evento de erro, na linha 37 e executa o método restartAfterError(), responsável por interromper o objeto Scanner através do método stop() e tentar se reconectar através do método execute(). A implementação dos métodos restartAfterError()e stop() é apresentada no Quadro 11.

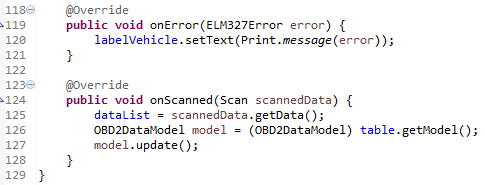
Quadro 11 - Métodos restartAfterError() e stop() da classe ObdJrpScanData



Fonte: Elaborado pelo autor.

Na linha 104 é disparada uma nova Thread que irá executar os métodos stop() e execute(), que podem ter uma execução demorada. Como o método restartAfterError() foi invocado a partir de um tratamento de evento, é disparada uma Thread para que o tratador de evento não permaneça bloqueado aguardando o fim da execução do stop() e execute().

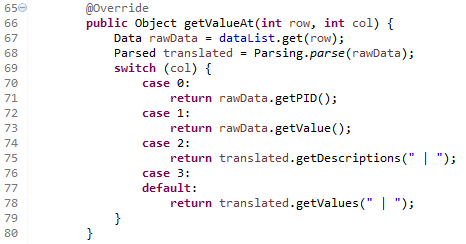
Quadro 12 - Atualizando interface de usuário em tempo real



Fonte: Elaborado pelo autor.

Na linha 125 é recebido o parâmetro scannedData, um objeto do tipo Scan, que contém as leituras obtidas pela interface ELM327, esse parâmetro é atribuído à variável de instância dataList e em seguida, nas linhas 126 e 127 é realizada a atualização de um objeto do tipo javax.swing.table.TableModel[[14]](#footnote-14) do qual parte relevante da implementação é apresentada no Quadro 13.

Quadro 13 - Apresentando dados lidos na interface de usuário



Fonte: Elaborado pelo autor.

Na linha 66 observa-se que o método getValueAt() recebe dois parâmetros, row e col, que correspondem aos índices da linha e coluna da célula a ser renderizada pelo componente visual javax.swing.JTable[[15]](#footnote-15). Na linha 67 utiliza-se o parâmetro row, para acessar um elemento Data do objeto dataList que foi inicializado na linha 125 do Quadro 12, e atribuir à variável rawData. Na linha 68 é utilizada a classe Parsing para transformar o objeto rawData em um objeto do tipo Parsed, atribuído à variável translated. Em seguida na instrução switch é tratado o parâmetro col de forma a presentar o valor apropriado na interface de usuário. Nas linhas 71 e 73 obtém-se respectivamente o PID e o valor em hexadecimal e nas linhas 75 e 78 obtém se a informação humanamente legível deste PID. Os valores renderizados podem ser observados na Figura 27, apresentada no início desta seção.

#### Comunicação com a interface ELM327 Bluetooth

[falar da classe ELM327]

#### Monitoramento dos dados pendentes de envio

[falar como é feito o monitoramento das leituras que não foram enviadas ao servidor]

#### Processamento das requisições no servidor

[falar como funciona o tratamento das requisições no servidor e geração dos gráficos]

### Operacionalidade da implementação

[Apresentação do funcionamento da implementação (em nível de usuário) através de um estudo de caso.]

## ANÁLISE DOS RESULTADOS

[Apresentar os casos de testes do software, destacando objetivo do teste, como foi realizada a coleta de dados e a apresentação dos resultados obtidos, preferencialmente em forma de gráficos ou tabelas, fazendo comentários sobre os mesmos.

Confrontar com os trabalhos correlatos apresentados na fundamentação teórica.]

# CONCLUSÕES

[As conclusões devem refletir os principais resultados alcançados, realizando uma avaliação em relação aos objetivos previamente formulados. Deve-se deixar claro se os objetivos foram atendidos, se as ferramentas utilizadas foram adequadas e quais as principais contribuições do trabalho para o seu grupo de usuários ou para o desenvolvimento científico/tecnológico.]

[Deve-se também incluir aqui as principais vantagens do seu trabalho e limitações.]

## EXTENSÕES

[Sugestões para trabalhos futuros.]

Referências

ALDERTON, L. Raspberry Pi - Bluetooth using Bluecove on Raspbian. **Luke Alderton**, p. 1, 3 jan 2015. Disponivel em: <http://lukealderton.com/blog/posts/2015/january/raspberry-pi-bluetooth-using-bluecove-on-raspbian.aspx>. Acesso em: 13 ago. 2016.

ANDROID PIT INTERNATIONAL. Android Apps - Lifestyle. **Android Pit International**, [S.l.], p. 1, 2016. Disponivel em: <https://www.androidpit.com/app/org.envirocar.app>. Acesso em: 20 maio 2016.

BLUECOVE. **BlueCove**, p. 1, 2008. Disponivel em: <http://www.bluecove.org>. Acesso em: 20 out. 2016.

CONAMA. Resolução CONAMA nº 354, de 13 de dezembro de 2004. Publicada no D.O.U. nº 239, de 14 de dezembro de 2004, Seção 1, p. 62-63, 2004. Disponivel em: <http://www.mma.gov.br/port/conama/legislacao/CONAMA\_RES\_CONS\_2004\_354.pdf>. Acesso em: 20 maio 2016.

ELM ELECTRONICS. OBD to RS232 Interpreter. **ELM Electronics - Circuits for the Hobbyist**, [S.l.], p. 1-94, 2016. Disponivel em: <http://www.elmelectronics.com/DSheets/ELM327DS.pdf>. Acesso em: 20 maio 2016.

ENVIROCAR. Off we go. **EnviroCar**, [S.l.], p. 1, 2015. Disponivel em: <http://envirocar.org>. Acesso em: 20 maio 2016.

GSM ASSOCIATION. Understanding the Internet of Things (IoT). **GSM Association**, [S.l.], p. 1, 2014. Disponivel em: <http://www.gsma.com/connectedliving/wp-content/uploads/2014/08/cl\_iot\_wp\_07\_14.pdf>. Acesso em: 20 maio 2016.

MANAVELLA, H. J. Diagnóstico Automotivo Avançado. **HM Autotrônica**, [S.l.], p. 121-127, 2009. Disponivel em: <http://www.hmautotron.eng.br/zip/cap19-hm004web.pdf>. Acesso em: 20 maio 2016.

NEW IT LIMITED. RPi 3 (2016) Model B. **New IT Limited**, [S.l.], p. 1, 2016. Disponivel em: <https://www.newit.co.uk/shop/all-raspberry-pi/raspberry\_pi\_3/raspberry\_pi3>. Acesso em: 20 maio 2016.

NG, A. The Evolution of the "Internet of Things": from "Diagnostics and Repair" to "Prescriptive and Proactive". **Horton Works**, [S.l.], p. 1, 2015. Disponivel em: <http://br.hortonworks.com/blog/the-evolution-of-the-internet-of-things-from-diagnostics-and-repair-to-prescriptive-and-proactive>. Acesso em: 20 maio 2016.

OUTILS OBD FACILE. Automotive Electronic Diagnostic. **Outils OBD Facile**, [S.l.], p. 1, 2015. Disponivel em: <http://www.outilsobdfacile.com/obd-mode-pid.php>. Acesso em: 20 maio 2016.

PYOBD. Open Source OBD2 Diagnostics. **OBD Tester**, [S.l.], p. 1, 2015. Disponivel em: <http://www.obdtester.com/pyobd>. Acesso em: 20 maio 2016.

RASPBERRY PI FOUNDATION. Teach, Learn and make with Raspberry Pi. **Raspberry Pi Foundation**, [S.l.], p. 1, 2016. Disponivel em: <https://www.raspberrypi.org>. Acesso em: 20 maio 2016.

RIORAND. On Board Diagnostics. **RioRand Advanced Technology**, [S.l.], p. 1, 2015. Disponivel em: <http://www.riorand.com/on-board-diagnostics>. Acesso em: 20 maio 2016.

SAE INTERNATIONAL. Society of Automotive Engineers. **DocumBase.com**, [S.l.], p. 1-228, 2006. Disponivel em: <http://www.documbase.com/goto/9552188-77674d95ff4a7fd13cf79446a8561c94/SAE-J1979-2006-edition-Ballot.pdf>. Acesso em: 20 maio 2016.

THE BEST OBD2 SCANNERS. 10 Modes of Operation for OBD2 Scanners. **The Best OBD2 Scanners**, [S.l.], p. 1, 2016. Disponivel em: <http://thebestobdiiscanners.com/10-modes-of-operation-for-obd2-scanners>. Acesso em: 20 maio 2016.

THOMPSON, T. J.; KLINE, P. J.; KUMAR, C. B. Bluetooth Application Programming with the Java APIs. Burlington: Morgan Kaufmann Publishers, 2008. p. 23-34. ISBN 978-0-12-374342-8. Disponivel em: <https://www.academia.edu/attachments/33201269/download\_file>. Acesso em: 20 out. 2016.

THOMSEN, A. Saiu o Raspberry Pi 3. **Filipe Flop Componentes Eletrônicos**, [S.l.], p. 1, 2016. Disponivel em: <http://blog.filipeflop.com/embarcados/saiu-o-raspberry-pi-3.html>. Acesso em: 20 maio 2016.

TOTAL CAR. ELM327 Review & About ELM 327 OBD2 Interface. **Total Car Diagnostics Support**, [S.l.], p. 1, 2014. Disponivel em: <http://www.totalcardiagnostics.com/support/Knowledgebase/Article/View/72/15/elm327-review--about-elm-327-obd2-interface>. Acesso em: 20 maio 2016.

ZAMBARDA, P. "Internet das Coisas": entenda o conceito e o que muda com a tecnologia. **Tech Tudo**, [S.l.], p. 1, 2014. Disponivel em: <http://www.techtudo.com.br/noticias/noticia/2014/08/internet-das-coisas-entenda-o-conceito-e-o-que-muda-com-tecnologia.html>. Acesso em: 20 maio 2016.

ZURAWSKI, R. Automative Embedded Systems Handbook. Boca Raton: CRC Press, 2009. p. 33-34. ISBN 978-0-8493-8026-6.

APÊNDICE A – Relação dos formatos das apresentações dos trabalhos

[Elemento opcional. **Apêndices são** **textos elaborados pelo autor** a fim de complementar sua argumentação. Os apêndices são identificados por letras maiúsculas consecutivas, seguidas de um travessão e pelos respectivos títulos. Deverá haver no mínimo uma referência no texto anterior para cada apêndice.]

[Colocar sempre um preâmbulo no apêndice. Não colocar tabelas e ou ilustrações sem identificação no apêndice. Caso existirem, identifique-as através da legenda, seguindo a numeração normal do volume final (para as legendas). Caso existirem tabelas e ou ilustrações, sempre referenciá-las antes.]

ANEXO A – Representação gráfica de contagem de citações de autores por semestre nos trabalhos de conclusões realizados no Curso de Ciência da Computação

[Elemento opcional. **Anexos são documentos não elaborados pelo autor**, que servem de fundamentação, comprovação ou ilustração, como mapas, leis, estatutos, entre outros. Os anexos são identificados por letras maiúsculas consecutivas, seguidas de um travessão e pelos respectivos títulos. Deverá haver no mínimo uma referência no texto anterior para cada anexo.]

[Colocar sempre um preâmbulo no anexo. Não colocar tabelas e ou ilustrações sem identificação no anexo. Caso existirem, identifique-as através da legenda, seguindo a numeração normal do volume final (para as legendas). Caso existirem tabelas e ou ilustrações, sempre referenciá-las antes.]

1. Horta (2000) explica que o sensor lambda é responsável pelo ajuste fino da mistura ar-combustível. [↑](#footnote-ref-1)
2. W3Schools (2016) explica que o método GET serve para requisitar dados de um determinado recurso e o método POST serve para enviar dados para serem processados por determinado recurso. [↑](#footnote-ref-2)
3. Rocha (2005, p. 52) explica que o padrão Singleton garante que uma classe só tenha uma única instância, e provê um ponto de acesso global a ela. [↑](#footnote-ref-3)
4. Segundo Rocha (2005), a padrão Observer define uma dependência um-para-muitos entre objetos para que quando um objeto mudar de estado, todos os seus dependentes sejam notificados e atualizados automaticamente. [↑](#footnote-ref-4)
5. Conforme Rocha (2005), Strategy permite que algoritmos mudem independentemente entre clientes que os utilizam. [↑](#footnote-ref-5)
6. SAE International (2006, p. 118-190) define os PIDs e respectivos algoritmos para obter informação legível. [↑](#footnote-ref-6)
7. Conforme SAE International (2006, p. 129), obtém-se as RPM, dividindo valor decimal dos bytes por 4. [↑](#footnote-ref-7)
8. Rocha (2005) explica que o padrão Façade define uma interface de nível mais elevado que torna o subsistema mais fácil de usar. [↑](#footnote-ref-8)
9. Segundo Rocha (2005), o padrão Command encapsula uma requisição na forma de um objeto, permitindo que clientes parametrizem diferentes requisições. [↑](#footnote-ref-9)
10. Repositório do BlueCove: http://repo1.maven.org/maven2/net/sf/bluecove/bluecove. [↑](#footnote-ref-10)
11. A API do trabalho correlato PyOBD poderia ser utilizado para o desenvolvimento em Python. [↑](#footnote-ref-11)
12. Documentação da classe UUID: http://www.bluecove.org/bluecove/apidocs/javax/bluetooth/UUID.html [↑](#footnote-ref-12)
13. VNC é um software de acesso remoto, disponível em https://www.realvnc.com/raspberrypi, que é disponibilizado gratuitamente com o sistema Raspbian, para fins não comerciais. [↑](#footnote-ref-13)
14. Documentação disponível em https://docs.oracle.com/javase/8/docs/api/javax/swing/table/TableModel.html. [↑](#footnote-ref-14)
15. Documentação disponível em https://docs.oracle.com/javase/8/docs/api/javax/swing/JTable.html. [↑](#footnote-ref-15)