

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:	<hr/> Prof. Miguel Alexandre Wisintainer, Mestre – Orientador, FURB
Membro:	<hr/> Prof(a). Nome do(a) Professor(a), Titulação – FURB
Membro:	<hr/> 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, mesmo perante todas as dificuldades, durante toda a vida.

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

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

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

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

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

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 Everaldo Artur Grahl, pela forma polida e objetiva com que lecionou e destacou a importância da engenharia e da qualidade de software.

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

Ao professor Dalton Solano dos Reis, que sempre soube tornar as aulas de computação gráfica e multimídia bastante interessantes e divertidas.

Ao professor Mauro Marcelo Mattos, pela objetividade com que conduzia as aulas e pelo entusiasmo em ajudar os alunos.

Ao amigo Stephan Dieter Biegling, pelo entusiasmo em testar o protótipo em seu veículo particular.

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.....	14
Figura 2 - Aspecto da interface ELM327 RS232	17
Figura 3 - Aspecto da interface ELM327 USB	17
Figura 4 - Aspecto da interface ELM327 Bluetooth	18
Figura 5 - Aspecto da interface ELM327 WiFi	18
Figura 6 - Blocos eletrônicos da interface ELM327.....	19
Figura 7 - Visão geral dos protocolos de comunicação OBD	19
Figura 8 - Características do Raspberry Pi 3 Model B	20
Figura 9 - Conectando PyOBD com o veículo	22
Figura 10 - Exibindo resultados de testes com PyOBD	22
Figura 11 - Verificando dados em tempo real com PyOBD	23
Figura 12 - Lendo e limpando códigos de falhas com PyOBD	23
Figura 13 - Velocidade do veículo no EnviroCar.....	24
Figura 14 - Velocidade média, trajeto e distância percorridos no EnviroCar	25
Figura 15 - Diversas informações coletadas pelo EnviroCar durante o percurso	25
Figura 16 - Ciclo de vida do firmware	28
Figura 17 - Ciclo de vida do servidor	29
Figura 18 - Camadas e pacotes do firmware	30
Figura 19 - Leitura de dados da interface ELM327 Bluetooth.....	31
Figura 20 - Transmissão dos dados pendentes	32
Figura 21 - Processamento de requisições no servidor.....	33
Figura 22 - Versões do Sistema Operacional e Java no Raspberry Pi.....	34
Figura 23 - Arquitetura da API JABWT BlueCove	34
Figura 24 - Executando ObdJrpListDevices no Raspberry Pi.....	41

LISTA DE QUADROS

Quadro 1 - Comparativo entre os trabalhos correlatos e o trabalho proposto	26
Quadro 2 - Listando dispositivos e serviços Bluetooth	36
Quadro 3 - Disparando consulta de dispositivos com JABWT	37
Quadro 4 - DiscoveryListener para consulta de dispositivos	37
Quadro 5 - Classe Lock utilizada para sincronização de processos	38
Quadro 6 - Disparando consulta de serviços com JABWT	39
Quadro 7 - DiscoveryListener para consulta de serviços	40

LISTA DE TABELAS

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

HDMI – High Definition Multimedia Interface

IOT – Internet of Things

JABWT – Java API for Bluetooth Wireless Technology

JNI – Java Native Interfaces

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.....	10
1.1	OBJETIVOS DO TRABALHO	11
1.2	ESTRUTURA.....	11
2	FUNDAMENTAÇÃO TEÓRICA	12
2.1	HISTÓRIA DO OBD	12
2.1.1	OBD NO BRASIL	12
2.2	OBD2.....	13
2.2.1	PROTOCOLOS OBD2	14
2.2.2	MODOS DE DIAGNÓSTICO.....	15
2.3	INTERFACE ELM327.....	16
2.4	RASPBERRY PI.....	20
2.5	TRABALHOS CORRELATOS	21
2.5.1	PYOBD	21
2.5.2	ENVIROCAR	23
2.5.3	COMPARATIVO ENTRE OS TRABALHOS	26
3	DESENVOLVIMENTO DO PROTÓTIPO	27
3.1	REQUISITOS.....	27
3.2	ESPECIFICAÇÃO	28
3.2.1	ESPECIFICAÇÃO DO FIRMWARE.....	29
3.2.2	ESPECIFICAÇÃO DO SERVIDOR	32
3.3	IMPLEMENTAÇÃO	33
3.3.1	Técnicas e ferramentas utilizadas.....	33
3.3.2	Operacionalidade da implementação	41
3.4	ANÁLISE DOS RESULTADOS	41
4	CONCLUSÕES.....	42
4.1	EXTENSÕES	42
	REFERÊNCIAS	43
	APÊNDICE A – RELAÇÃO DOS FORMATOS DAS APRESENTAÇÕES DOS	
	TRABALHOS	45

ANEXO A – REPRESENTAÇÃO GRÁFICA DE CONTAGEM DE CITAÇÕES DE AUTORES POR SEMESTRE NOS TRABALHOS DE CONCLUSÕES REALIZADOS NO CURSO DE CIÊNCIA DA COMPUTAÇÃO.....	46
--	-----------

1 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 análises 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.

1.1 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:

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

1.2 ESTRUTURA

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

2 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 0 apresenta a plataforma Raspberry Pi e, por fim, na seção 2.5 são descritos dois trabalhos correlatos.

2.1 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).

2.1.1 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:

- a) sensor de pressão absoluta ou fluxo de ar;
- b) sensor de posição da borboleta;
- c) sensor de temperatura de arrefecimento;
- d) sensor de temperatura de ar;
- e) sensor de oxigênio;
- f) sensor de velocidade do veículo;
- g) sensor de posição do eixo comando de válvulas;
- h) sensor de posição do virabrequim;
- i) sistemas de recirculação dos gases de escape;
- j) sensor para detecção de detonação;

- k) válvulas injetoras;
- l) sistema de ignição;
- m) módulo controle eletrônico do motor;
- n) lâmpada indicadora de mau funcionamento; e
- o) outros componentes que o fabricante julgue relevantes para a correta avaliação do funcionamento do veículo e controle de emissões de poluentes.

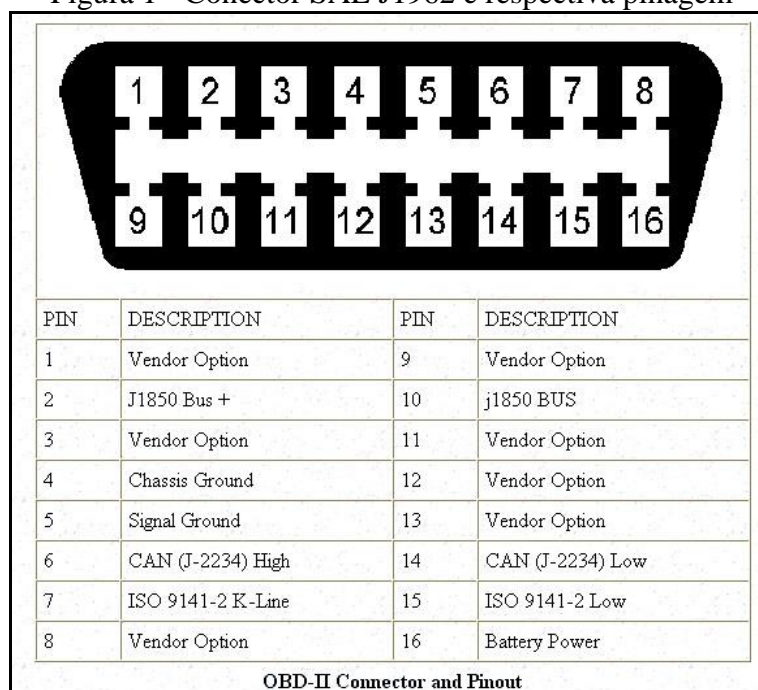
CONAMA (2004) considera que o sistema OBDBr-2 complementa as funções e características do sistema OBDBr-1. Segundo CONAMA (2004), o sistema OBDBr-2 deve detectar e registrar a existência de falhas, deterioração dos sensores de oxigênio e eficiência de conversão do catalisador. CONAMA (2004) complementa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

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

2.2 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). Os diversos parâmetros que podem ser consultados são denominados Parameter Identification Numbers (PID) e os códigos de erros são denominados Diagnostic Trouble Codes (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.

2.2.1 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.

2.2.1.1 REDES CAN

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

- a) ISO 157565: Utilizado por todos os veículos. Velocidade de comunicação de 125 a 500 Kbps;

- b) SAE J1939: Utilizado principalmente por veículos pesados como caminhões e máquina agrícolas. Velocidade de comunicação de 125 a 500 Kbps.

2.2.1.2 LINHAS K/L

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

- a) ISO 9141-2: Utilizado principalmente por fabricantes europeus. Velocidade de comunicação de 10,4 Kbps;
- b) 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.

2.2.1.3 SAE J1850

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

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

2.2.2 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).

- a) 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;
- b) modo 2: Obtém o “instantâneo” de uma falha. Quando a ECU detecta uma falha, ela grava os dados do sensor daquele momento específico;
- c) 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:
 - P0xxx: Para falhas associadas ao motor e transmissão,
 - C0xxx: Para falhas associadas ao chassi,

- B0xxx: para falhas associadas à carroceria,
 - U0xxx: para falhas associadas à comunicação de rede;
- d) modo 4: Utilizado para apagar os DTCs gravados e desligar o MIL;
 - e) modo 5: Retorna o autodiagnostico do sensor lambda. Segundo Outils OBD Facile (2015), este modo não é mais utilizado pois o modo 6 substitui suas funções;
 - f) modo 6: Retorna os resultados do autodiagnostico realizado nos diversos sensores do veículo;
 - g) 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;
 - h) 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;
 - i) modo 9: Este modo obtém informações do veículo como por exemplo seu número de identificação;
 - j) 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.

2.3 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:

- a) ELM327 RS232: Conexão serial que está gradativamente desaparecendo nos computadores modernos. A Figura 2 apresenta o aspecto desta interface;
- b) ELM327 USB: Conexão Universal Serial Bus (USB), presente na maioria dos computadores atuais. A Figura 3 apresenta o aspecto desta interface;
- c) ELM327 Bluetooth: Conexão sem fio, que pode ser utilizada com computadores ou smartphones. A Figura 4 apresenta o aspecto desta interface;
- d) 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



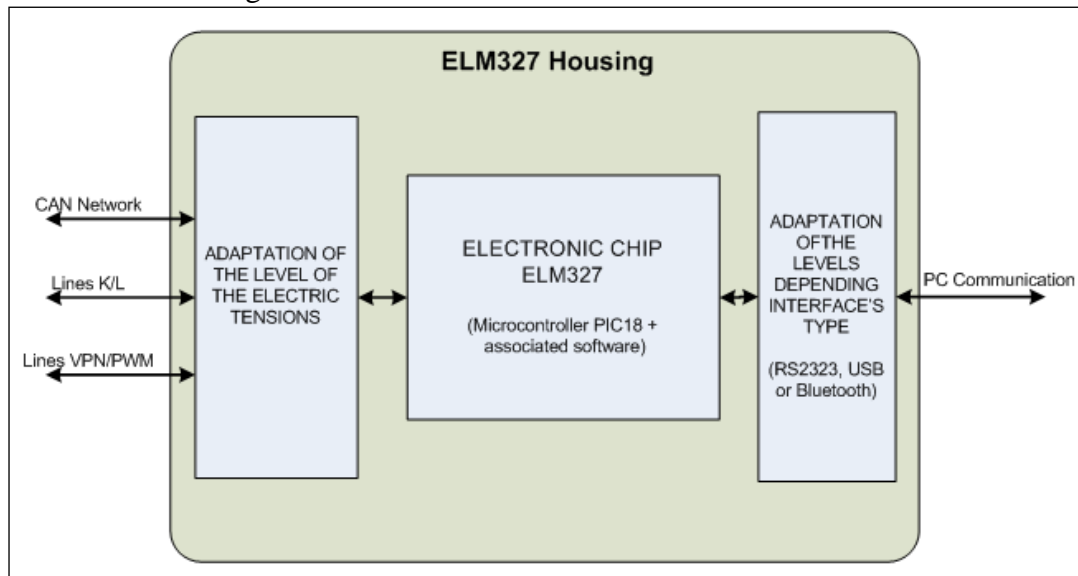
Fonte: Total Car (2014).

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:

- a) 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;

- b) 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;
- c) 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.

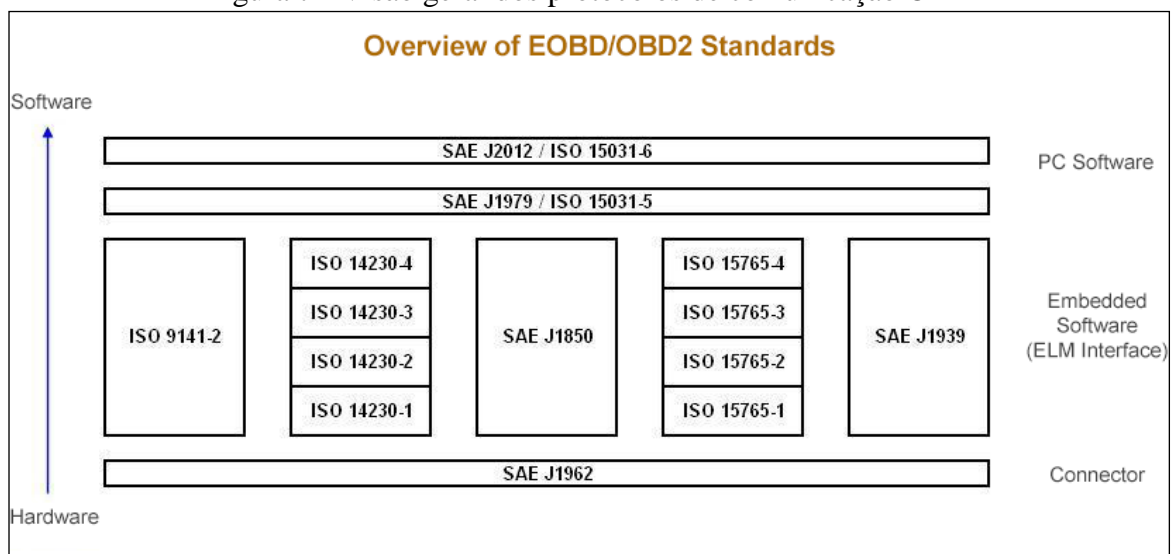
Figura 6 - Blocos eletrônicos da interface ELM327



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

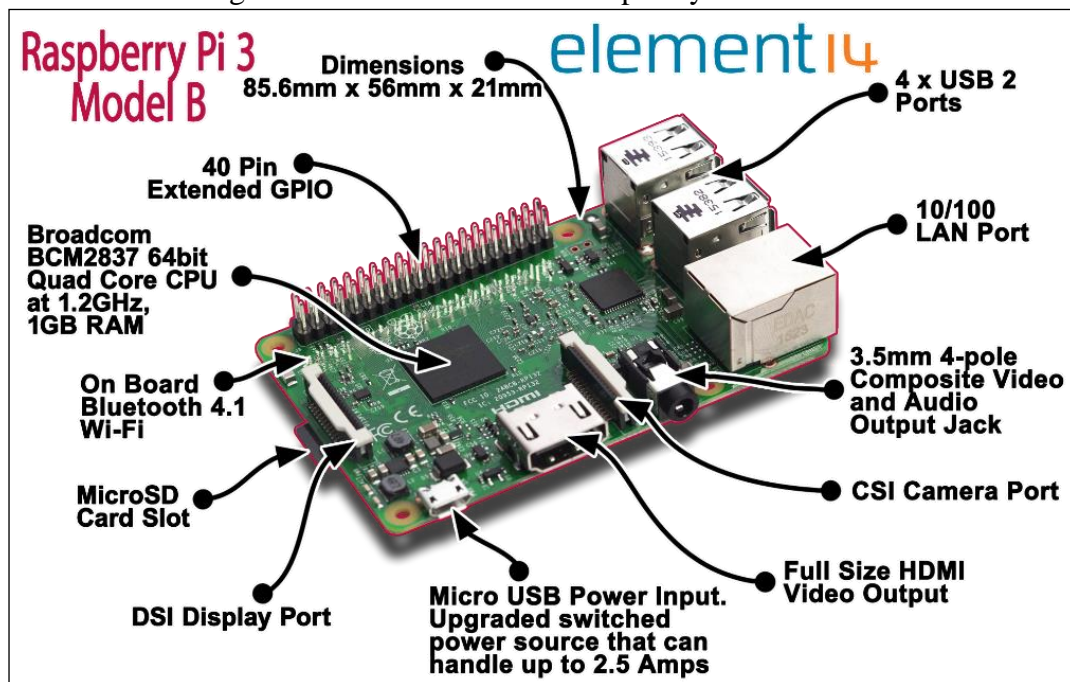


Fonte: Total Car (2014).

2.4 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:

- computador de placa única com chipset Broadcom BCM2837;
- processador quad core ARM Cortex-A53 de 1,2GHz;
- 1GB de Random Access Memory (RAM);
- 40 pinos de General Purpose Input/Output (GPIO);
- conexão Bluetooth 4.1 integrada;
- conexão WiFi 802.11n integrada;
- 1 porta Ethernet 10/100;
- 4 portas USB;
- 1 conector de 4 polos, combinado para saída de áudio estéreo e vídeo composto;
- 1 saída High Definition Multimedia Interface (HDMI);

- k) 1 porta Camera Serial Interface (CSI);
- l) 1 porta Display Serial Interface (DSI);
- m) 1 porta micro SanDisk (SD), para carga do sistema operacional e armazenamento de dados;
- n) 1 porta micro USB para fonte de alimentação.

2.5 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.

2.5.1 PYOBD

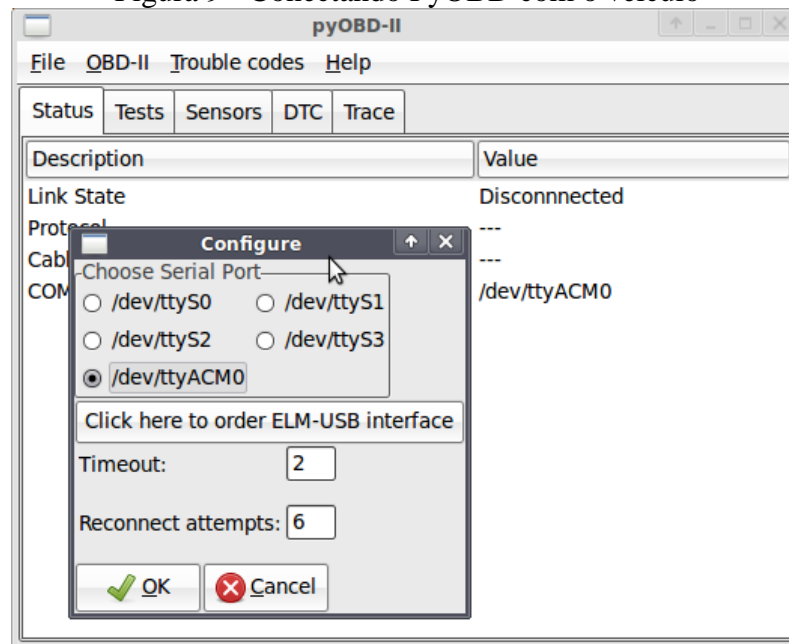
Trata-se de uma ferramenta open source de diagnóstico automotivo, segundo PyOBD (2015), a ferramenta foi projetada para se conectar à porta OBD2 através de uma interface ELM327 USB. PyOBD é voltado para desenvolvedores Python, é composto de um único módulo, chamado `obd_io`, que permite um controle de 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:

- a) uma interface ELM327 USB;
- b) python 2.x ou superior;
- c) pacote `py_serial`;
- d) um veículo que implemente o padrão OBD2.

Com o PyOBD é possível:

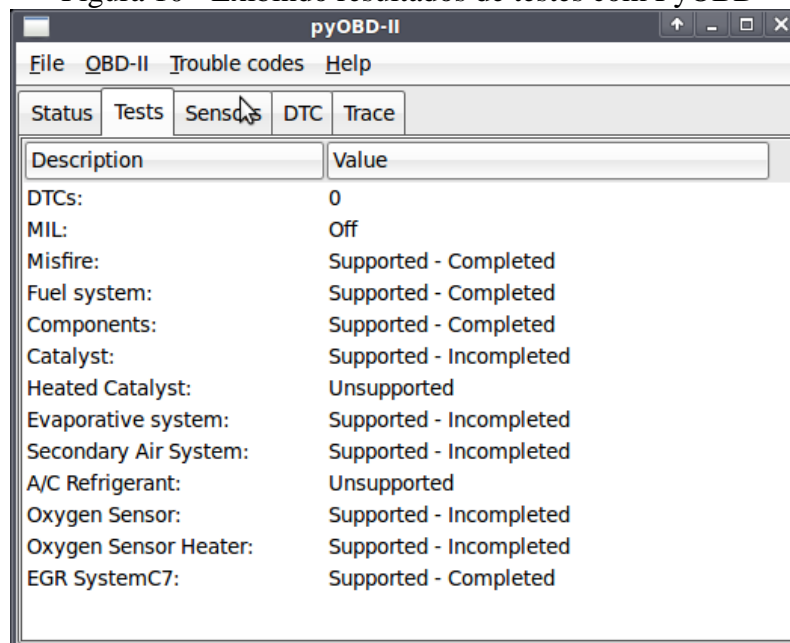
- a) conectar-se ao veículo, conforme ilustrado na Figura 9;
- b) exibir resultados de testes, conforme Figura 10;
- c) verificar dados dos sensores em tempo real, conforme Figura 11;
- d) ler e limpar códigos de falhas DTC, conforme Figura 12.

Figura 9 - Conectando PyOBD com o veículo



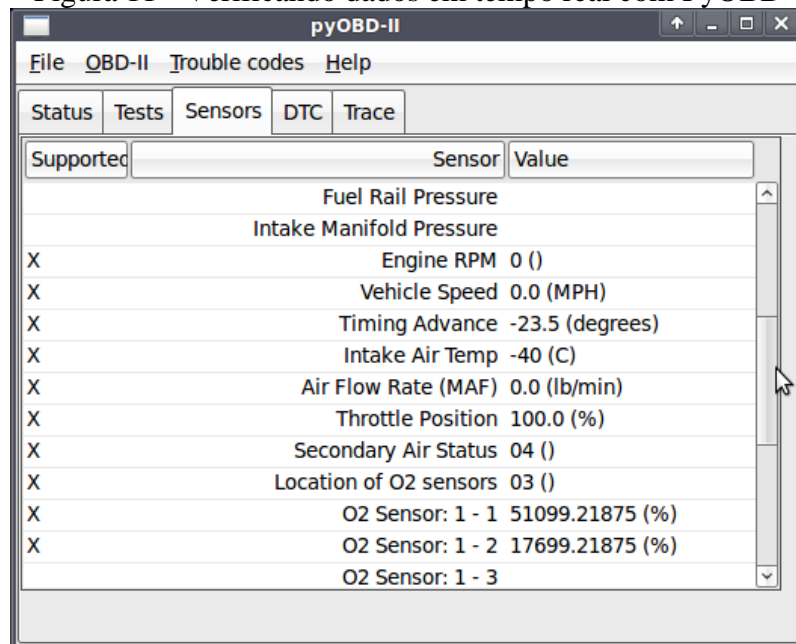
Fonte: PyOBD (2015).

Figura 10 - Exibindo resultados de testes com PyOBD



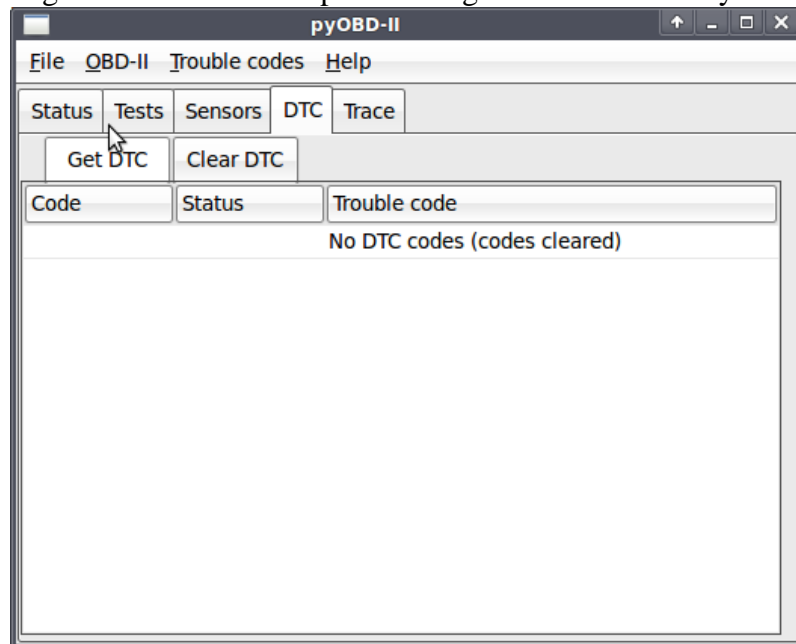
Fonte: PyOBD (2015).

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).

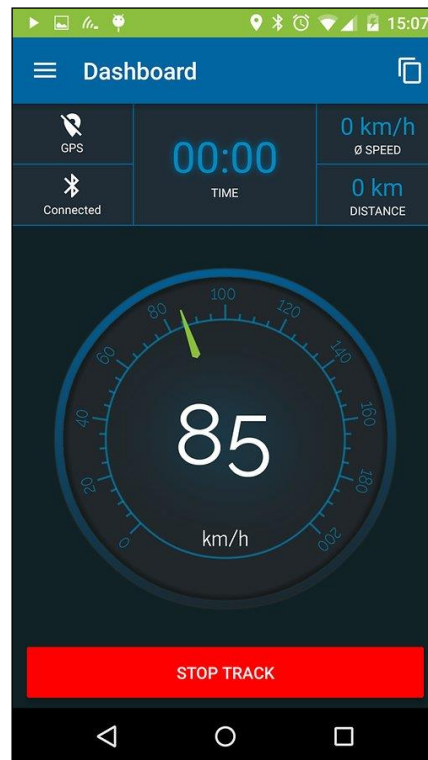
2.5.2 ENVIROCAR

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 carbônico e de ruídos (ANDROID PIT INTERNATIONAL, 2016).

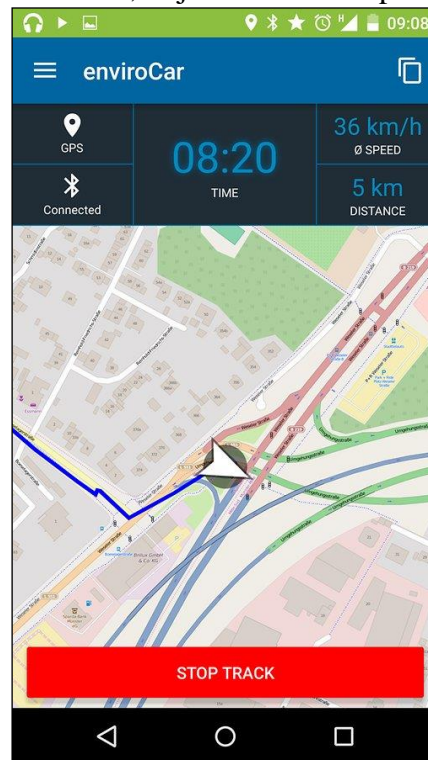
A seguir algumas telas do aplicativo EnviroCar. Na Figura 13 observa-se a velocidade do veículo em quilômetros por hora. A Figura 14 apresenta um mapa com o desenho do trajeto percorrido, o tempo da viagem, a distância percorrida e a velocidade média. Na Figura 15 são apresentadas diversas informações 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



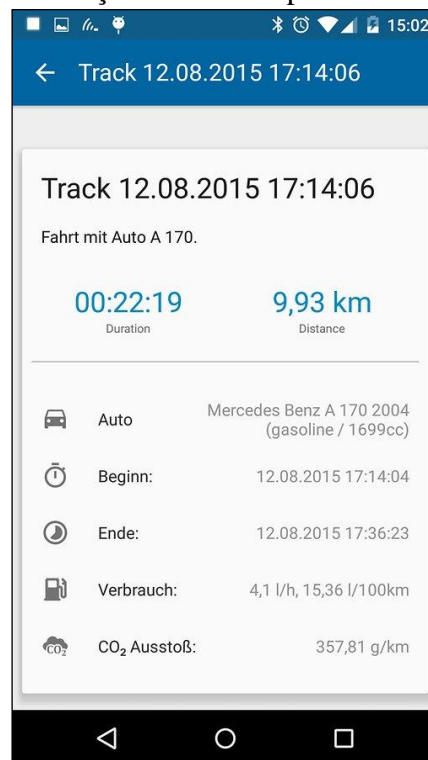
Fonte: Android Pit International (2016).

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



Fonte: Android Pit International (2016).

Figura 15 - Diversas informações coletadas pelo EnviroCar durante o percurso



Fonte: Android Pit International (2016).

2.5.3 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		X	X
Disponibiliza gráficos dos dados coletados			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.

3 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.

3.1 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:

- a) o firmware deverá ser inicializado automaticamente ao ligar a placa Raspberry Pi (Requisito Funcional – RF);
- b) o firmware deverá se conectar à porta OBD2 através de uma interface ELM327 Bluetooth (RF);
- c) o firmware deverá coletar os dados da porta OBD2 e armazená-los localmente até serem enviados ao servidor (RF);
- d) o firmware deve tentar estabelecer uma conexão com o servidor a cada 5 minutos, caso não esteja conectado à internet (RF);
- e) 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);
- f) o firmware deverá ser desenvolvido utilizando tecnologia Java SE (Requisito Não Funcional – RNF);
- g) o firmware deverá executar em sistema operacional Raspbian (RNF);
- h) o servidor deverá responder requisições HTTP, através dos métodos get e post (RF);
- i) o servidor deverá persistir os dados coletados pelo firmware (RF);
- j) o servidor deverá persistir os dados em arquivos XML, sem a necessidade de utilizar banco de dados (RNF);
- k) o servidor deverá dispor uma página web para consultar os dados OBD2 a partir do número do chassi do carro (RF).
- l) o servidor deverá ser desenvolvido utilizando tecnologia Java EE (RNF);
- m) o servidor deverá executar no servidor de aplicações Apache TomCat (RNF);
- n) a página web deve apresentar gráficos com os valores dos dados coletados (RF);
- o) a página web deve apresentar uma tabela com os valores dos dados coletados (RF);
- p) a página web deve ter interface responsiva de modo que possa ser visualizada em

smartphones (RNF);

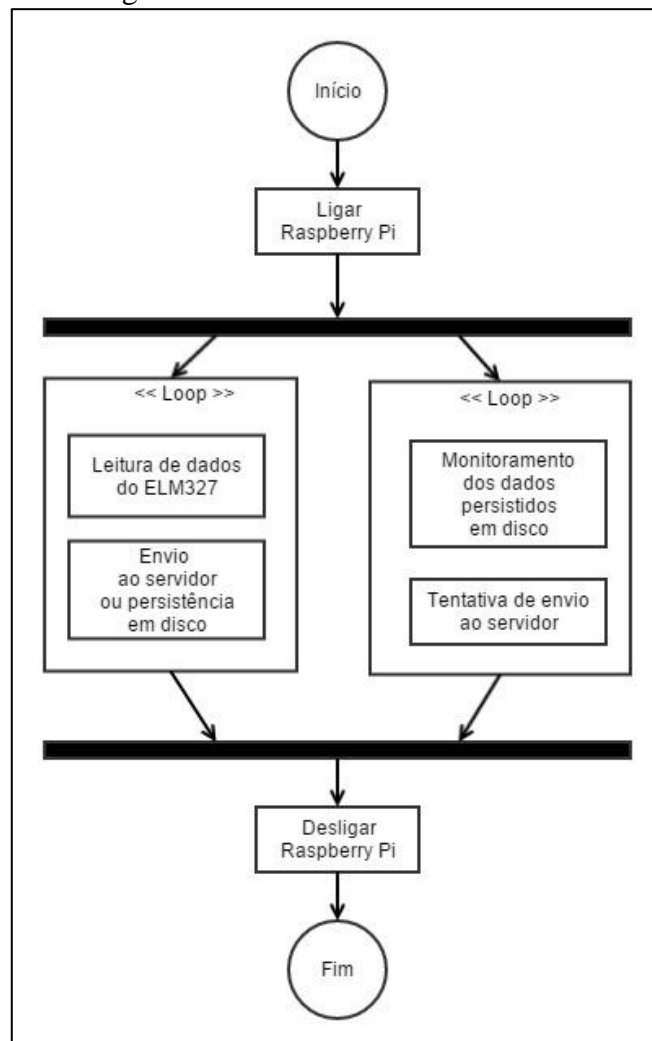
q) a página web deverá ser desenvolvida utilizando HTML, CSS e JavaScript (RNF).

3.2 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.

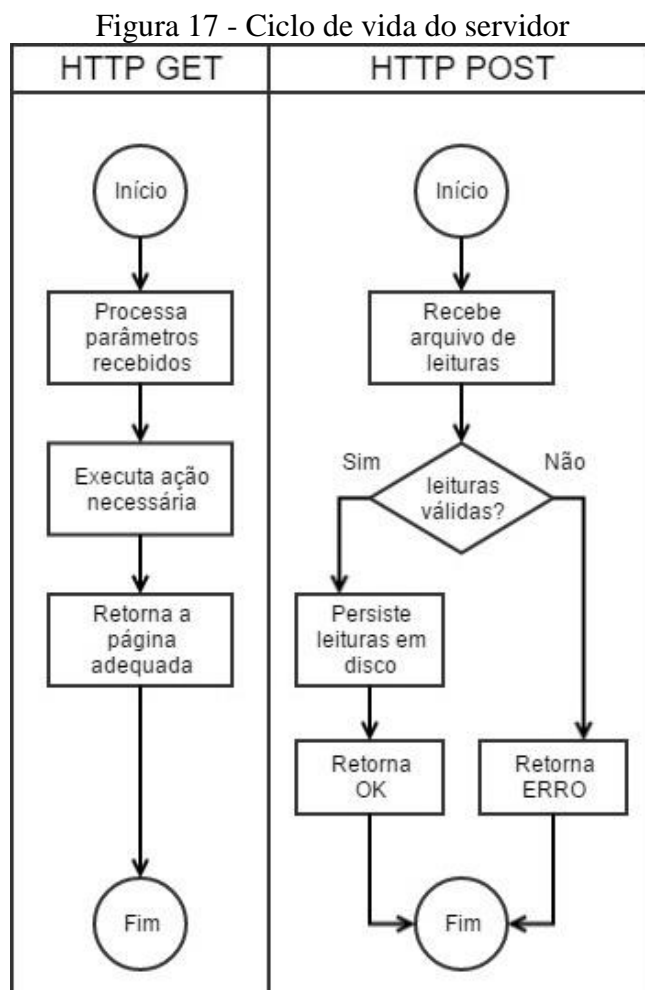
O ciclo de vida do firmware consiste em disparar dois processos paralelos após a inicialização do Raspberry Pi, um processo é responsável pela leitura dos dados da interface ELM327 Bluetooth e o outro processo é responsável por monitorar o diretório onde os pacotes com os dados das leituras são persistidos.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

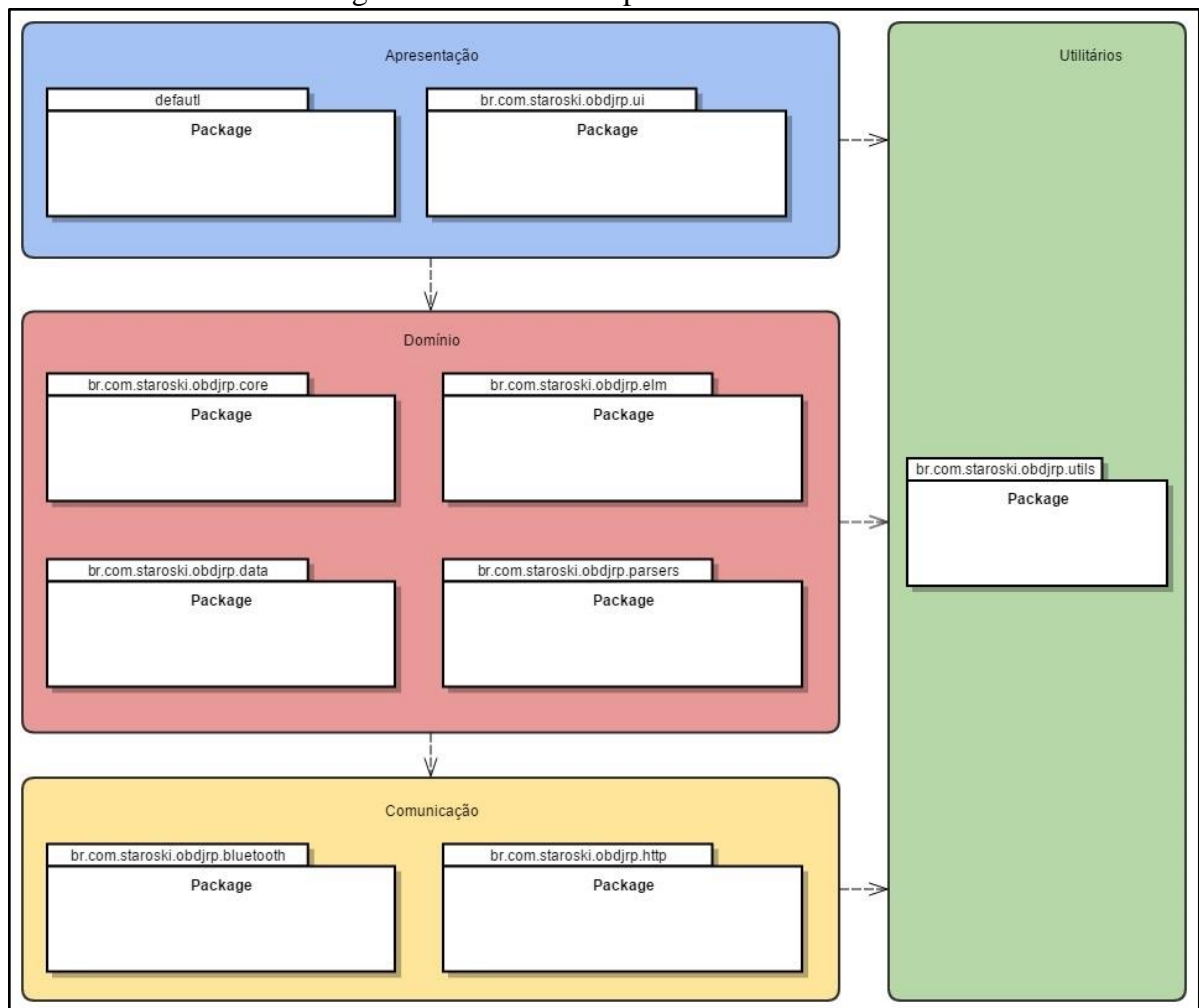


Fonte: Elaborado pelo autor.

3.2.1 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 e HTTP 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.

3.2.1.1 LEITURA DE DADOS DA INTERFACE ELM327 BLUETOOTH

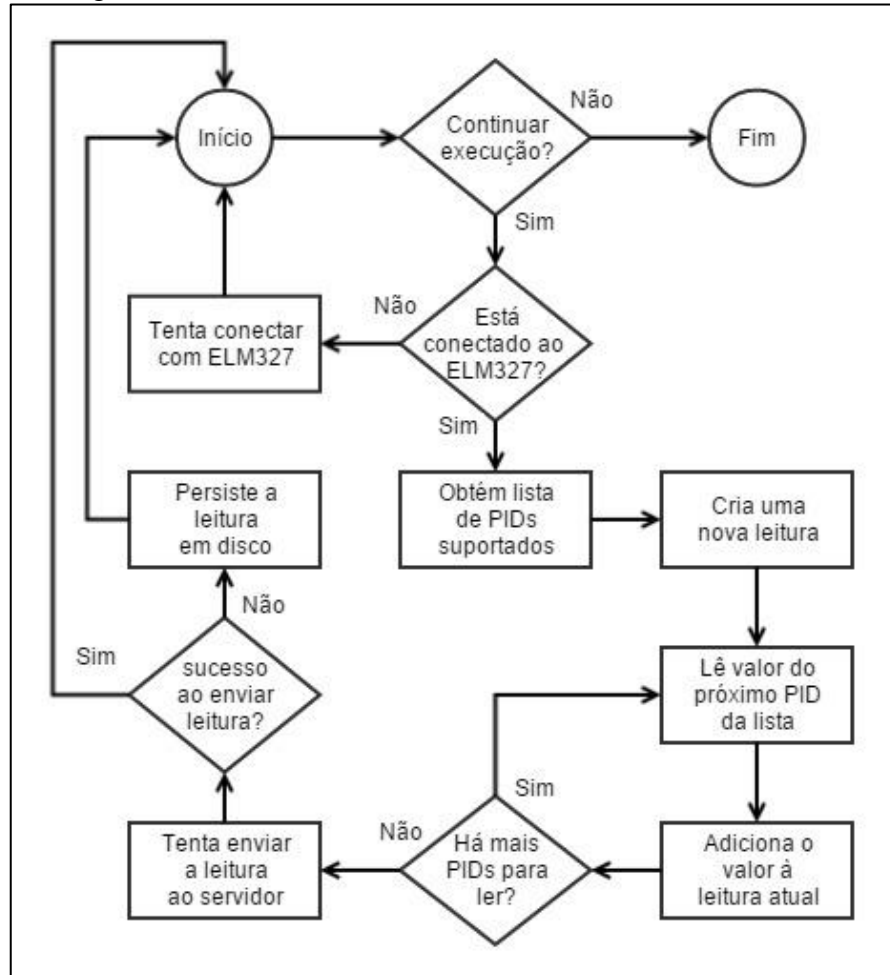
Conforme citado na seção 3.2, o ciclo de vida do firmware consiste na execução de dois processos paralelos, o primeiro destes processos especificado, é a leitura dos dados da interface ELM327 Bluetooth. A leitura destes dados consiste em um laço que realiza as seguintes operações:

- a) se estiver conectado ao ELM327, executa o passo c), senão executa o passo b);
- b) tenta conectar-se ao ELM327 e volta ao passo a);
- c) solicita ao ELM327 a lista dos PIDs suportados pelo veículo e executa o passo d);
- d) cria um objeto de leitura para armazenar os valores dos PIDs e executa o passo d);
- e) obtém o valor do próximo PID suportado e executa o passo f);
- f) adiciona o valor do PID lido ao objeto de leitura e executa o passo g);
- g) se houver mais PIDs para ler, volta ao passo e), senão executa o passo h);
- h) tenta enviar o objeto de leitura ao servidor e executa o passo i);

- i) se conseguiu enviar o objeto de leitura, volta ao passo a), senão executa o passo j);
- j) persiste o objeto de leitura em disco e volta ao passo a).

Na Figura 19 observa-se o fluxo do o processo de leitura da interface ELM327 Bluetooth.

Figura 19 - Leitura de dados da interface ELM327 Bluetooth



Fonte: Elaborado pelo autor.

3.2.1.2 TRANSMISSÃO DOS DADOS PENDENTES

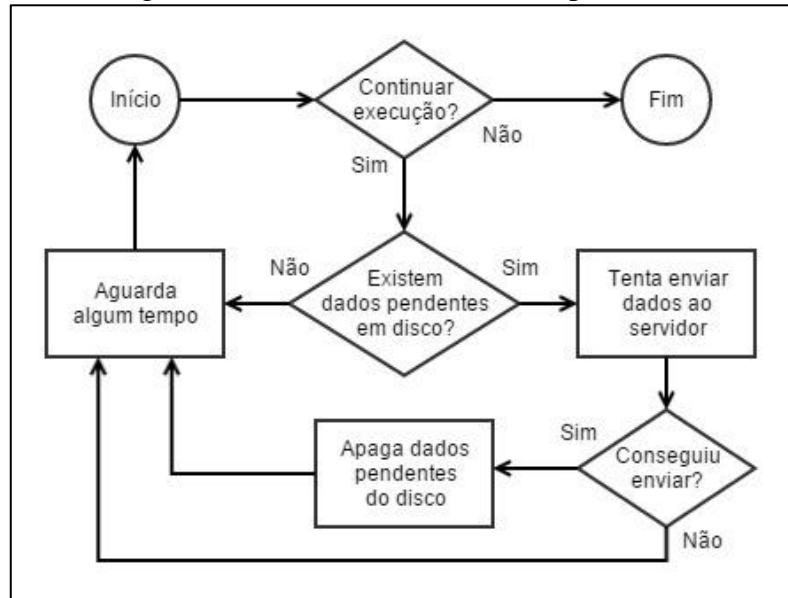
O segundo processo do ciclo de vida do firmware é a transmissão dos dados pendentes, este processo trata de monitorar o diretório onde o primeiro processo persistiu os objetos de leitura que não foram enviados ao servidor. Assim como a leitura dos dados da interface ELM327 Bluetooth, a transmissão dos dados pendentes também consiste em um laço que realiza as seguintes operações:

- a) verifica se há arquivos pendentes no diretório, se houver, executa o passo c), senão executa o passo b);
- b) aguarda 5 minutos e volta ao passo a);

- c) tenta enviar cada arquivo pendentes o servidor e segue ao passo d);
- d) apaga do disco, cada arquivo enviado com sucesso ao servidor, e volta ao passo b).

Na Figura 20 observa-se o fluxo do processo de transmissão dos dados pendentes.

Figura 20 - Transmissão dos dados pendentes

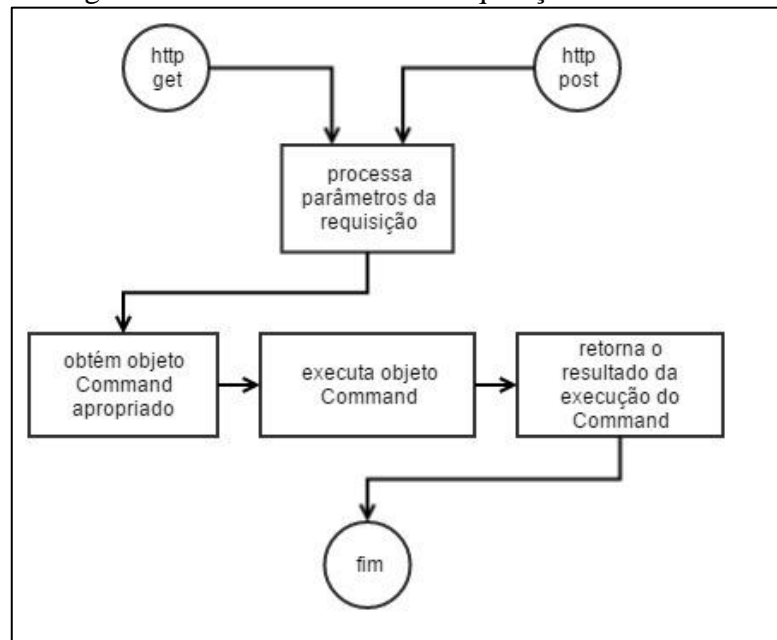


Fonte: Elaborado pelo autor.

3.2.2 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 `get` quanto requisições `post`. Para processar as requisições, utilizou-se o padrão de projeto Command, de forma que, a partir dos parâmetros recebidos, se obtenha um objeto apropriado para tratar a requisição. Na Figura 21 observa-se como ocorre o fluxo de processamento de requisições no servidor.

Figura 21 - Processamento de requisições no servidor



Fonte: Elaborado pelo autor.

3.3 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.

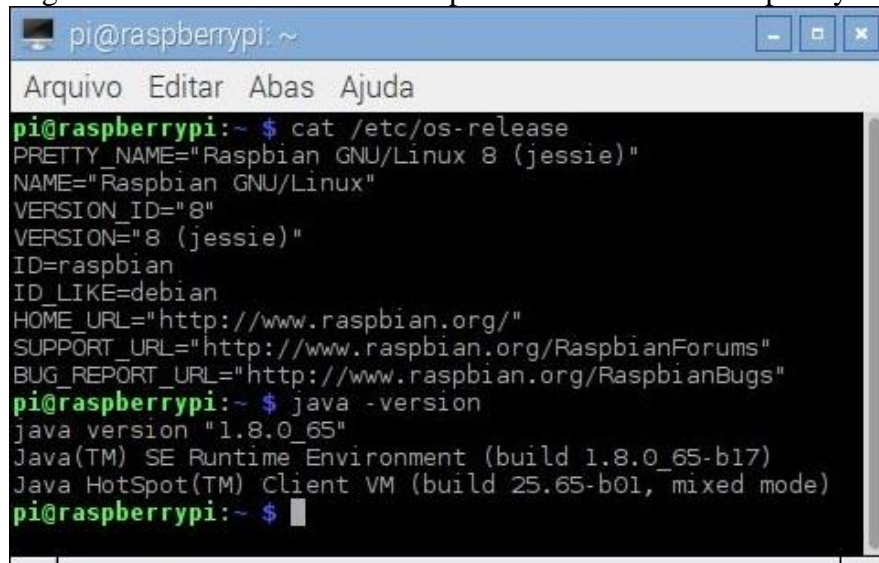
3.3.1 Técnicas e ferramentas utilizadas

As implementações tanto do firmware quanto do servidor, foram realizadas utilizando o ambiente de desenvolvimento Eclipse Neon com linguagem de programação Java. Para o desenvolvimento do firmware foi utilizada 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 utilizada 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.

3.3.1.1 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 22.

Figura 22 - Versões do Sistema Operacional e Java no Raspberry Pi



```

pi@raspberrypi: ~
Arquivo  Editar  Abas  Ajuda

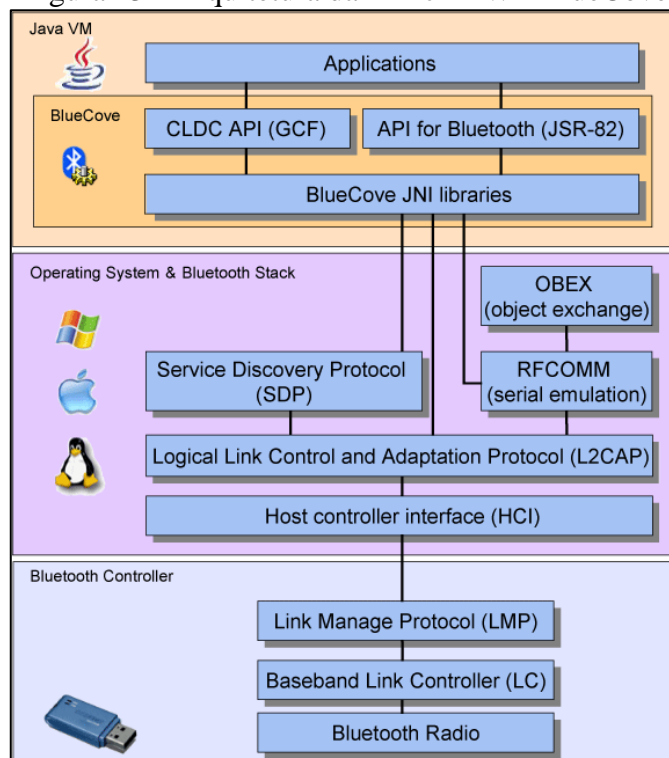
pi@raspberrypi:~ $ cat /etc/os-release
PRETTY_NAME="Raspbian GNU/Linux 8 (jessie)"
NAME="Raspbian GNU/Linux"
VERSION_ID="8"
VERSION="8 (jessie)"
ID=raspbian
ID_LIKE=debian
HOME_URL="http://www.raspbian.org/"
SUPPORT_URL="http://www.raspbian.org/RaspbianForums"
BUG_REPORT_URL="http://www.raspbian.org/RaspbianBugs"
pi@raspberrypi:~ $ java -version
java version "1.8.0_65"
Java(TM) SE Runtime Environment (build 1.8.0_65-b17)
Java HotSpot(TM) Client VM (build 25.65-b01, mixed mode)
pi@raspberrypi:~ $

```

Fonte: Elaborado pelo autor.

Conforme citado na seção 2.4, o Raspberry Pi funciona de forma análoga à um desktop 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 Interfaces (JNI) para a comunicação via Bluetooth, trocando mensagens diretamente com os drivers do sistema operacional, como pode ser observado na arquitetura apresentada na Figura 23.

Figura 23 - Arquitetura da API JABWT BlueCove



Fonte: BlueCove (2008).

Observou-se que no repositório 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:

- a) instalar as ferramentas Maven e Ant no Raspberry Pi;
- b) executar o comando Maven para criar os diretórios de código fonte compatíveis com o ambiente Eclipse:


```
- mvn eclipse:clean eclipse:eclipse -DdownloadSources=true;
```
- c) executar o comando Ant para compilar as bibliotecas nativas e gerar o arquivo JAR da biblioteca Java:


```
- 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, que também é disponibilizada com o Raspian GNU/Linux 8.

3.3.1.2 Listando dispositivos Bluetooth no firmware

A primeira aplicação desenvolvida para o firmware foi um programa em linha de comando para listar 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 listar os serviços disponibilizados por cada dispositivo.

Quadro 2 - Listando dispositivos e serviços Bluetooth

```

11 public final class ObdJrpListDevices {
12
13     public static void main(String[] args) {
14         try {
15             ObdJrpListDevices program = new ObdJrpListDevices();
16             program.execute();
17         } catch (Throwable t) {
18             t.printStackTrace();
19             System.exit(-1);
20         }
21     }
22
23     private void execute() throws IOException {
24         List<RemoteDevice> devices = Bluetooth.listDevices();
25         for (RemoteDevice device : devices) {
26             String address = device.getBluetoothAddress();
27             String name = device.getFriendlyName(false);
28             System.out.printf("device \"%s\" - \"%s\" {", address, name);
29             printServices(device);
30             System.out.printf("}\n\n");
31         }
32     }
33
34     private void printServices(RemoteDevice device) throws IOException {
35         List<ServiceRecord> services = Bluetooth.listServices(device);
36         for (ServiceRecord service : services) {
37             System.out.printf("\n\t\t \"%s\" ", Bluetooth.getServiceName(service));
38         }
39     }
40 }

```

Fonte: Elaborado pelo autor.

Observa-se na linha 24 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, tradução nossa), 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

```

141 public static List<RemoteDevice> listDevices() throws IOException {
142     synchronized (LOCK) {
143         DiscoveryAgent agent = LocalDevice.getLocalDevice().getDiscoveryAgent();
144         agent.cancelInquiry(DEVICE_LISTENER);
145         DEVICE_LISTENER.reset();
146         if (agent.startInquiry(DiscoveryAgent.GIAC, DEVICE_LISTENER)) {
147             LOCK.lock();
148         }
149     }
150     return DEVICE_LISTENER.getDevices();
151 }

```

Fonte: Elaborado pelo autor.

Observa-se 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

```

22 private static class DeviceDiscovery extends DiscoveryAdapter {
23
24     private final List<RemoteDevice> devices = new LinkedList<>();
25
26     @Override
27     public void deviceDiscovered(RemoteDevice btDevice, DeviceClass cod) {
28         devices.add(btDevice);
29     }
30
31     public List<RemoteDevice> getDevices() {
32         return devices;
33     }
34
35     @Override
36     public void inquiryCompleted(int discType) {
37         LOCK.unlock();
38     }
39
40     public void reset() {
41         devices.clear();
42     }
43 }
44
45 private static final DeviceDiscovery DEVICE_LISTENER = new DeviceDiscovery();

```

Fonte: Elaborado pelo autor.

Na linha 28 do `DiscoveryListener`, o dispositivo recebido pelo parâmetro `btDevice` é adicionado à lista `devices` e na linha 37, quando a consulta termina, é invocado o método `LOCK.unlock()`, de forma a desbloquear o método `Bluetooth.listDevices()` e devolver 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 checkada `InterruptedException`, forçando o desenvolvedor a tratar ou relançar a exceção. Os métodos da classe `Lock` criada, não declaram o lançamento de nenhuma exceção checkada, e o método `lock()` trata a `InterruptedException` e a transforma em uma exceção não checkada 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

```

3 public final class Lock {
4
5     private final Object LOCK = new Object();
6
7     public void lock() {
8         lock(0);
9     }
10
11     public void lock(long timeout) {
12         try {
13             synchronized (LOCK) {
14                 LOCK.wait(timeout);
15             }
16         } catch (InterruptedException e) {
17             throw new RuntimeException("Lock interrupted!", e);
18         }
19     }
20
21     public void unlock() {
22         synchronized (LOCK) {
23             LOCK.notifyAll();
24         }
25     }
26 }

```

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, identificado através do valor hexadecimal `0100` e o UUID correspondente ao perfil Serial Port Profile (SPP) identificado através do valor hexadecimal `0000110100001000800000805F9B34FB`. 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

```

154 public static List<ServiceRecord> listServices(RemoteDevice device) throws IOException {
155     synchronized (LOCK) {
156         int[] attributes = new int[] { Bluetooth.NAME };
157         UUID[] uuids = new UUID[] { Bluetooth.SPP };
158         DiscoveryAgent agent = LocalDevice.getLocalDevice().getDiscoveryAgent();
159         agent.cancelServiceSearch(SERVICE_LISTENER.getTransactionID());
160         SERVICE_LISTENER.reset();
161         int id = agent.searchServices(attributes, uuids, device, SERVICE_LISTENER);
162         SERVICE_LISTENER.setTransactionID(id);
163         if (id > 0) {
164             LOCK.lock();
165         }
166     }
167     return SERVICE_LISTENER.getServices();
168 }

```

Fonte: Elaborado pelo autor.

Observa-se que caso a invocação do `searchServices()`, retorne um `id` maior que zero, é invocado o método `LOCK.lock()`, este método faz 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, sendo também 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

```

48 private static class ServiceDiscovery extends DiscoveryAdapter {
49
50     private final List<ServiceRecord> services = new LinkedList<>();
51
52     private int transactionID;
53
54     public List<ServiceRecord> getServices() {
55         return services;
56     }
57
58     public int getTransactionID() {
59         return transactionID;
60     }
61
62     public void reset() {
63         services.clear();
64     }
65
66     @Override
67     public void servicesDiscovered(int transID, ServiceRecord[] servRecord) {
68         for (ServiceRecord service : servRecord) {
69             services.add(service);
70         }
71     }
72
73     @Override
74     public void serviceSearchCompleted(int transID, int respCode) {
75         LOCK.unlock();
76     }
77
78     public void setTransactionID(int transactionID) {
79         this.transactionID = transactionID;
80     }
81 }
82
83 private static final ServiceDiscovery SERVICE_LISTENER = new ServiceDiscovery();

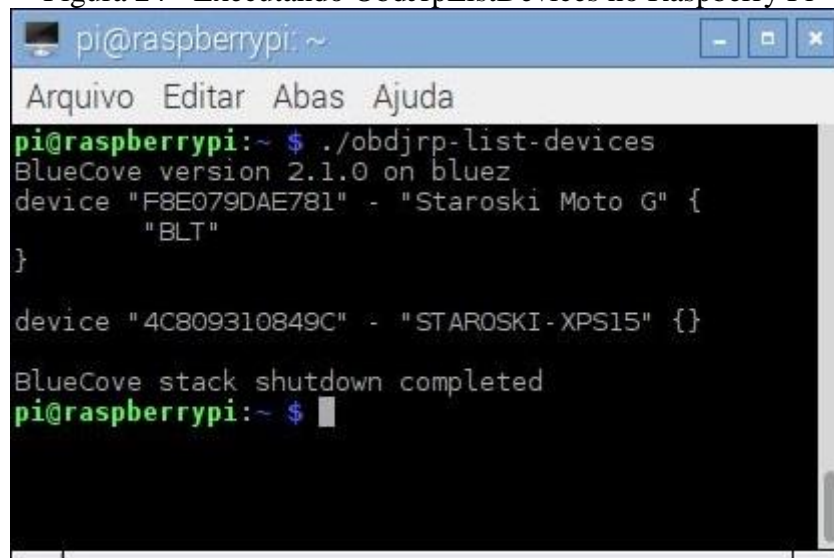
```

Fonte: Elaborado pelo autor.

Na linha 69 do DiscoveryListener, cada serviço descoberto recebido pelo parâmetro `servRecord` é adicionado à lista `services` e na linha 75, quando a consulta termina, é invocado o método `LOCK.unlock()`, de forma a desbloquear o método `Bluetooth.listServices()` e devolver a lista de serviços descobertos para o dispositivo informado.

Na Figura 24 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 24 - Executando ObdJrpListDevices no Raspberry Pi



```

pi@raspberrypi: ~
Arquivo Editar Abas Ajuda
pi@raspberrypi:~ $ ./obdjrp-list-devices
BlueCove version 2.1.0 on bluez
device "F8E079DAE781" - "Staroski Moto G" {
    "BLT"
}

device "4C809310849C" - "STAROSKI-XPS15" {}

BlueCove stack shutdown completed
pi@raspberrypi:~ $

```

Fonte: Elaborado pelo autor.

3.3.1.3 Comunicação com a interface ELM327 Bluetooth

[falar sobre a como foi implementado a comunicação com o adaptador OBD2]

3.3.1.4 Monitoramento dos dados pendentes de envio

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

3.3.1.5 Processamento das requisições no servidor

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

3.3.2 Operacionalidade da implementação

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

3.4 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.]

4 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.]

4.1 EXTENSÕES

[Sugestões para trabalhos futuros.]

REFERÊNCIAS

- ALDERTON, L. Raspberry Pi - Bluetooth using Bluecove on Raspbian. **Luke Alderton**, p. 1, 3 jan 2015. Disponível 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. Disponível em: <<https://www.androidpit.com/app/org.envirocar.app>>. Acesso em: 20 maio 2016.
- BLUECOVE. **BlueCove**, p. 1, 2008. Disponível 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. Disponível 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. Disponível em: <<http://www.elmelectronics.com/DSheets/ELM327DS.pdf>>. Acesso em: 20 maio 2016.
- ENVIROCAR. Off we go. **EnviroCar**, [S.l.], p. 1, 2015. Disponível em: <<http://envirocar.org>>. Acesso em: 20 maio 2016.
- GSM ASSOCIATION. Understanding the Internet of Things (IoT). **GSM Association**, [S.l.], p. 1, 2014. Disponível 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. Disponível 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. Disponível 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. Disponível 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. Disponível 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. Disponível 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. Disponível em: <<https://www.raspberrypi.org>>. Acesso em: 20 maio 2016.
- RIORAND. On Board Diagnostics. **RioRand Advanced Technology**, [S.l.], p. 1, 2015. Disponível 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. Disponível 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. Disponível 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. Disponível 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. Disponível 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. Disponível 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. Disponível 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. Automotive 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.]