UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE SÃO CARLOS

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Sistema Autônomo de Vigilância Baseado em Dados Biológicos com Registro de Dados na Nuvem via *Smartphone*

Autores: Enzo Bertini Vieira e Lara Bertini Vieira

Orientador: Prof. Dr. Evandro L. L. Rodrigues

São Carlos

2014

ENZO BERTINI VIEIRA e LARA BERTINI VIEIRA

Sistema Autônomo de Vigilância Baseado em Dados Biológicos com Registro de Dados na Nuvem via *Smartphone*

Trabalho de Conclusão de Curso apresentado
à Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia Elétrica

ORIENTADOR: Prof. Dr. Evandro L. L. Rodrigues

São Carlos
2014

Vieira, Enzo Bertini
V658s Sistema autônomo de vigilância baseado em dados biológicos com registro de dados na nuvem via *Smartphone* / Enzo Bertini Vieira, Lara Bertini Vieira; orientador Evandro L.L.Rodrigues. -- São Carlos, 2014.

Monografia (Graduação em Engenharia Elétrica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2014.

1. Biometria. 2. Autonomia. 3. Bluetooth. 4. Banco de dados. 5. Computação na nuvem. 6. Android. I. Titulo. II. Vieira, Lara Bertini.

FOLHA DE APROVAÇÃO

Nome: Enzo Bertini Vieira

Título: "Arquitetura de Hardware e Software para sistema de vigilância baseado no conceito de nuvem de dados"

Trabalho de Conclusão de Curso defendido e aprovado em 27 / 06 / 2014,

com NOTA 10,0 (Dez, zero), **pela Comissão Julgadora:**

Prof. Associado Evandro Luís Linhari Rodrigues - (Orientador - SEL/EESC/USP)

Prof. Associado Diógenes Pereira Gonzaga - (Professor Aposentado - SEL/EESC/USP)

Prof. Dr. Marcelo Andrade da Costa Vieira - (SEL/EESC/USP)

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Homero Schiabel

FOLHA DE APROVAÇÃO

Nome: Lara Bertini Vieira

Título: "Arquitetura de Hardware e Software para sistema de vigilância baseado no conceito de nuvem de dados"

Trabalho de Conclusão de Curso defendido e aprovado em 27/06/2014,

com NOTA 10,0 (Dez, zero), **pela Comissão Julgadora:**

Prof. Associado Evandro Luís Linhari Rodrigues - (Orientador - SEL/EESC/USP)

Prof. Associado Diógenes Pereira Gonzaga - (Professor Aposentado - SEL/EESC/USP)

Prof. Dr. Marcelo Andrade da Costa Vieira - (SEL/EESC/USP)

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Homero Schiabel

Dedicatória

Mais do que este trabalho, dedico todo meu empenho diário, desde a formação básica até o título de engenheiro, a minha família, em especial aos meus pais, irmã, tia, avó e namorada. Através do cuidado, do apoio, do amor, do carinho, das críticas e dos conselhos que vocês me proporcionam, sempre tenho forças para traçar e atingir minhas metas e objetivos.

Enzo Bertini Vieira

Dedico este TCC e todo esforço depositado em seu desenvolvimento aos integrantes de minha família, que sempre disponibilizaram-se a apoiar-me e amar-me mesmo discordando de certas decisões.

Lara Bertini Vieira

Agradecimentos

Agradeço a minha irmã e parceira de TCC, Lara, por ter sido a pessoa que mais conviveu comigo durante minha vida acadêmica, sempre se disponibilizando a ajudar no que fosse necessário; a minha tia Isabel e avó Olga, por sempre estarem preocupadas com meu bem estar, acompanhando de perto todo meu crescimento pessoal e acadêmico; a minha namorada Roberta, por estar sempre presente ao meu lado, apoiando, equilibrando, amando e me completando de modo a me tornar um ser humano melhor e mais feliz; aos meus colegas de curso, professores e funcionários por ajudar com as dificuldades intrínsecas de um curso de engenharia, auxiliando na resolução de problemas e entendimento das matérias; por fim, agradeço em especial aos meus pais, Ângela e Givaldo, por nunca terem economizado esforços em proporcionar uma forte base de ensino, educação e confiança, possibilitando-me ingressar numa universidade de qualidade, ensinando-me a tomar minhas próprias decisões e a aprender com suas consequências.

Enzo Bertini Vieira

Sou muito grata a meus pais e familiares, cuja educação, críticas e conversas francas resultaram na construção consciente de meu caráter e mentalidade como pessoa. Sou grata aos professores, funcionários e amigos que durante meus anos na USP me aconselharam e me trataram bem. Sou grata especialmente aos professores Diógenes Gonzaga, Evandro Rodrigues e Rodrigo Ramos.

Lara Bertini Vieira

"Don’t only practice your art, but force your way into its secrets,
for it and knowledge can raise men to the divine"
(Ludwig van Beethoven)

Resumo

Este projeto visa uma nova abordagem para um sistema de vigilância móvel autônomo. Uma pulseira biométrica coleta dados sobre o batimento cardíaco do usuário e os envia, via *bluetooth*, para o módulo de processamento, um *smartphone* com sistema operacional Android, que faz a análise destes batimentos cardíacos e define se o usuário está passando por alguma situação de risco de morte, ou extremo estresse. O *smartphone* escreve, periodicamente, informações em tempo imediato sobre o usuário, como localização geográfica e batimentos cardíacos, num banco de dados na nuvem que, por sua vez, já contém informações pessoais do usuário, como o tipo sanguíneo, telefone para contato emergencial, alergias, entre outros. A premissa é que esses dados na nuvem sejam acessíveis a todo o momento por um servidor externo para servirem de base para um socorro médico emergencial, ou seja, o sistema visa prover auxílio imediato de uma maneira proativa e autônoma, de modo a preservar vidas, sendo também de baixo custo e alcançável a grande parte da população. Foram realizados diversos testes e simulações demonstrando a adequabilidade do sistema proposto para as situações-problema apresentadas.

Palavras-Chave: biometria, autonomia, *bluetooth*, banco de dados, computação na nuvem, Android.

Abstract

This project seeks a new approach to an autonomous and mobile surveillance system. A biometric wristband collects data about the user's heartbeat and sends them via bluetooth to the processing module, a smartphone with Android operational system, which analyzes the heart rate and determines whether the user is undergoing some death risk, or extreme stress. The smartphone writes information, periodically and in immediate-time, about the user, such as geographical location and heartbeat, in a cloud database service, which in turn, already contain personal user information such as blood type, contact phone emergency, allergies, among others. The premise is that data in the cloud are constantly accessible by an external server as a basis for emergency medical help, in other words, the system aims to provide immediate assistance in a proactive and autonomous way, in order to preserve life, being low-cost and accessible to the majority of the population. Several tests and simulations demonstrate the suitability of the proposed system to solve the problems presented.

Keywords: biometrics, autonomy, bluetooth, database, cloud computing, Android.

Lista de Figuras

Lista de Tabelas

Siglas

ABESE	Associação Brasileira das Empresas de Sist. Eletrônicos de Segurança
AD	*Analog to Digital* - Analógico para Digital
ADT	*Android Development Tools* - Ferramentas para Desenvolv. Android
API	*Application Programming Interface* - Interface Programação Aplicativos
app	*Application* - Aplicativo
AVD	*Android Virtual Device* - Dispositivo Virtual do Android
BD	Banco de Dados
CPF	Cadastro de Pessoa Física
DC	*Direct Current* - Corrente Contínua
DPI	*Dots per Inch* - Pontos por Polegada
DVM	*Dalvik Virtual Machine* - Máquina Virtual Dalvik
GPRS	*General Packet Radio Service* - Serviço de Rádio de Pacote Geral
GPS	*Global Positioning System* - Sistema de Posicionamento Global
ID	*Identification* - Identificação
IDE	*Integrated Development Environment* - Ambiente Integrado de Desenvolv.
I/O	*Input/Output* - Entrada/Saída
JDK	*Java Development Kit* - Kit de Desenvolvimento Java
JSON	*JavaScript Object Notation* - Notação de Objeto JavaScript
JVM	*Java Virtual Machine* - Máquina Virtual Java
MP	Módulo de Processamento de Dados e Envio de Alerta
ODGM	*Object Database Management Group* - Grupo Gestão de BDs a Objetos
PB	Pulseira biométrica
PDA	*Personal Digital Assistant* - Assistente Pessoal Digital
PC	*Personal Computer* - Computador Pessoal

PIC	*Peripherical Interface Controller* - Interface Controladora de Periféricos
POO	Programação Orientada a Objetos
SGBD	Sistemas Gerenciadores de Bancos de Dados
SQL	*Structured Query Language* - Linguagem de Consulta Estruturada
SO	Sistema Operacional
UI	*User Interface* - Interface do Usuário
UML	*Unified Modeling Language* - Linguagem de Modelagem Unificada
URL	*Uniform Resource Locator* - Endereço de Recurso Disponível na Rede
XML	*Extensible Markup Language* - Linguagem de Marcação Extensível

Sumário

Capítulo 1

Introdução

1.1 Segurança

De acordo com a ABESE, no Brasil, existem mais de 12 mil empresas atuando no segmento de sistemas eletrônicos de segurança, contudo apenas 9% são fabricantes, os outros dividem-se entre revendedores e instaladores, monitoradores, integradores e distribuidores [3].

Em 2010, esse setor movimentou US$ 1,680 bilhões, apresentando um crescimento de 12% em comparação ao ano anterior. Nos últimos nove anos, esse mercado vem crescendo com taxa média anual de 13%, gerando cerca de 125 mil empregos diretos e mais de 1,4 milhão indiretos [3].

Contudo, apesar do aumento no oferecimento de serviços e sistemas de segurança, nota-se também grande crescimento na taxa de roubos, vandalismo, incêndio, assaltos, furtos de carga, extorsão a profissionais e empresas, e estupros, sendo que pelo menos 80% da população brasileira já foi vítima de violência, direta ou indiretamente [4]. Esse paradigma é decorrente, em grande parte, à inexistência de um sistema realmente confiável e autônomo.

Roubo de cargas nas rodovias brasileiras, por exemplo, é uma realidade que cresce 7% ao ano: somente em 2009 o Brasil teve um prejuízo aproximado de R$ 1 bilhão de reais (cerca de 13.500 ocorrências)[5]. Da mesma forma, o aumento no índice de roubos de cargas tem prejudicado exportadores de café da região do Triângulo Mineiro. Nos últimos meses foram, pelo menos, 10 caminhões roubados, com estimativa de perda de 2,5 bilhões de reais. O fato de que 60% das mercadorias que circulam no Brasil utilizam vias rodoviárias torna o problema ainda mais agravante [5].

Consequentemente, observa-se morte de motoristas e vigilantes, prejuízos psicológicos, financeiros e grandes intranquilidades no ambiente de negócios de logística em nosso país.

Vale notar que não basta apenas a melhoria de infraestrutura do sistema de segurança, mas é imprescindível uma integração maior entre as polícias rodoviária, civil e militar, para que a informação do roubo flua rapidamente e o auxílio seja efetivo [6].

A opinião pública brasileira segue a tendência do mercado. Em 1989, 15% da população preocupava-se com a segurança pública. Em 2012 este valor cresceu para 42%, sendo que 51% da população classificou a segurança pública como ruim ou péssima [4] e [7].

Visando uma solução para esses problemas de segurança, foi proposto a elaboração de um sistema seguro, confiável e, acima de tudo, autônomo, isentando assim o usuário da responsabilidade de indicar a presença de uma situação de risco.

1.2 Requisitos Do Sistema

Baseado num estudo sobre as tecnologias existentes no que tange aos sistemas de segurança pessoal e corporativo, verificou-se que, atualmente os sistemas de segurança não possuem autonomia o suficiente para garantir uma resposta emergencial independente do usuário. Tal prática não é eficiente, uma vez que o usuário pode estar impossibilitado de acionar o sistema, ou pode até mesmo não conseguir fazê-lo, seja por medo, ansiedade ou nervosismo.

Foi constatado também que os sistemas atuais apenas servem para "remediar"a situação, não atuando sobre evitar a situação de risco, nem de preservar o usuário. Dentre esses sistemas, os mais comuns para segurança móvel são os "bloqueadores"(apenas para veículos), que consistem num sistema que bloqueia a ignição dos veículos após o mesmo ser dado como roubado, os "localizadores"que, mediante solicitação, informa a posição geográfica do bem furtado, e os "Rastreadores"que guardam um histórico de localização do bem furtado. Desta maneira, todos eles não garantem a segurança do indivíduo, bem como necessitam de uma intervenção humana para que se identifique uma situação como de risco, e seus custos são muito altos, como pode ser visto na tabela 1.1 [2].

Tipo	Custo Médio de Instalação	Custo Médio de Manutenção Mensal
Bloqueador	1.000 reais	40 reais
Localizador	1.000 reais	60 reais
Rastreador	6.000 10.000 reais	250 reais

Tabela 1.1: Custo dos Sistemas de Segurança [2]

Desta maneira, mostrou-se necessário o desenvolvimento de uma ferramenta que pudesse integrar um *hardware* de alta qualidade com um *software* de alta confiabilidade, de modo que

tanto a identificação das situações de risco quanto sua correspondente ajuda devessem ser realizadas automaticamente (sem que houvesse a necessidade do usuário acionar um botão de emergência, por exemplo), reduzindo também o tempo de resposta a tais situações.

Contudo, um dos desafios baseava-se no fato de encontrar um método para acionar o sistema automaticamente, ou seja, seria necessário pesquisar uma maneira confiável de se detectar uma situação de risco, e tal detecção deveria ocorrer no menor tempo possível.

1.3 Método de Acionamento do Alarme

Buscando a autonomia no acionamento do alarme, optou-se por coletar os batimentos cardíacos do usuário do sistema de segurança, e, a partir do mesmo, verificar a existência ou não de uma situação de risco. Tal opção se mostrou viável pois o corpo humano possui respostas conhecidas quando exposto à situações de estresse.

Nos capítulos posteriores, serão abordadas informações mais detalhadas sobre o porque deste método de acionamento do alarme, bem como um melhor detalhamento da resposta biológica em situações de estresse.

1.4 Definição do Sistema em Alto Nível

Tendo em mente os problemas de segurança já abordados, bem como os requisitos de projeto, pensou-se em soluções práticas para o problema em questão. Desta maneira, após diversos estudos críticos, propõe-se nesse trabalho um sistema composto por dois módulos, sendo eles o módulo de aquisição dos batimentos cardíacos, chamado neste trabalho de pulseira biométrica (PB), e o módulo de processamento de dados e envio de alerta (MP).

A título de comodidade, estética e conforto de utilização, decidiu-se por fazer a comunicação entre os dois módulos de maneira sem fio, via bluetooth, uma vez que seria muito incômodo a utilização dos dois módulos acoplados ao pulso do usuário.

Como definiu-se que o sistema de detecção de situação de perigo seria feito tomando-se como base a variação dos batimentos cardíacos, definiu-se a necessidade de um transdutor adequado para a coleta desta propriedade. Com a aquisição do sinal realizada, o mesmo deve ser processado e enviado para o MP que, por sua vez, deve verificar a existência ou não de uma situação de perigo para o usuário. Após a verificação, as informações relativas ao usuário, como batimento cardíaco, posição geográfica, entre outras devem ser escritas num

banco de dados na nuvem.

Como o batimento cardíaco é uma variável individual, ou seja, cada pessoa possui um batimento cardíaco próprio, dependendo de seu estado de atividade corporal (repouso ou exercício), o algoritmo para avaliação dos batimentos cardíacos e detecção de perigo deve ter um comportamento de modo a reconhecer tais estados, para poder tomar decisões corretas sobre a presença ou não de uma situação de risco. Maiores detalhes sobre a arquitetura do sistema e dos algoritmos serão apresentadas nos próximos capítulos.

1.5 Objetivos

O objetivo consiste no desenvolvimento de um sistema de segurança autônomo, modular e portátil, baseado na resposta cardíaca do corpo humano em reações de perigo. Tanto o *hardware* como o *software* do sistema possui como principal pilar a confiabilidade e rapidez para uma comunicação em tempo imediato. O projeto foi voltado para a área de segurança, e tem como responsabilidade social a redução de criminalidade, roubos e mortes por falha de segurança.

Capítulo 2

Especificação do Projeto

Este capítulo visa esclarecer e descrever o funcionamento do sistema desenvolvido, incluindo a interação entre a PB e o MP, transmitindo assim o escopo do projeto. Todo embasamento teórico, descrição dos materiais utilizados e os métodos serão tema dos capítulos seguintes.

2.1 Funcionamento do Sistema

Como já explicado de maneira genérica no capítulo 1, o sistema contará com dois módulos (PB e MP), cuja comunicação é feita sem fio, identificando situações de risco de maneira autônoma, dada a resposta cardíaca ao medo e ao estresse.

A interação entre estes dois módulos se dará via *bluetooth*. Como o banco de dados utilizado será na nuvem, a comunicação entre o MP e o Banco de Dados se dará pela internet. A figura 2.1 apresenta um ilustrativo para tal funcionamento.

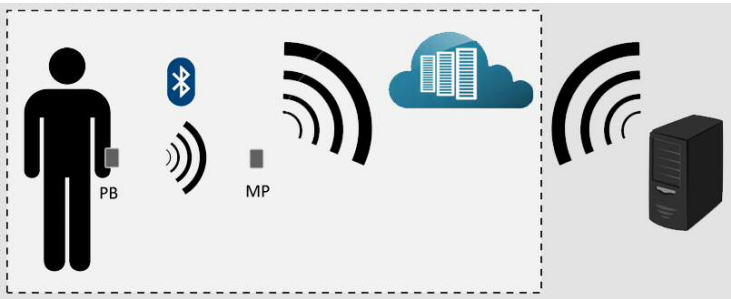

Figura 2.1: Esquema de Funcionamento do Sistema

Ressalta-se que este trabalho de conclusão de curso se limita a implementação até a etapa de população do Banco de Dados (conforme hachurado na figura 2.1). Optou-se por não desenvolver a parte do servidor, visando manter o foco no dispositivo móvel e sua integração

com um *hardware* externo.

2.2 Pulseira Biométrica

A pulseira biométrica contém um transdutor que coletará informações referentes ao batimento cardíaco do usuário. Este sensor, por sua vez, tem sua resposta enviada para um circuito para processar o sinal o quanto necessário de modo a torná-lo utilizável por um microprocessador. O microprocessador contabiliza os batimentos cardíacos por minuto e estes são disponibilizados via *bluetooth* para o MP.

A figura 2.2 apresenta o fluxograma de operação da PB

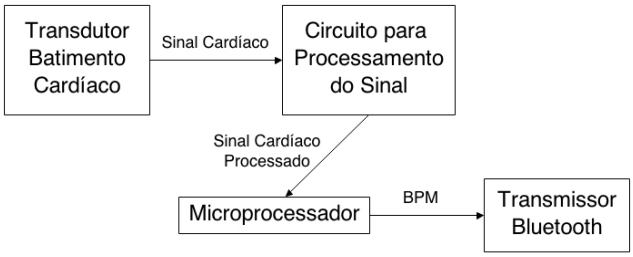

Figura 2.2: Fluxograma de Operação da PB

2.3 Módulo de Processamento

Por definição de projeto, o banco de dados deve conter, além do batimento cardíaco do usuário, sua localização (latitude e longitude). Após um estudo constatou-se que a melhor maneira de implementar tal módulo seria utilizando um *smartphone* com sistema operacional Android, uma vez que o mesmo possui em sua estrutura circuitos de comunicação *bluetooth*, de localização geográfica e acesso à internet móvel.

A figura 2.3 apresenta o fluxograma de operação do MP.

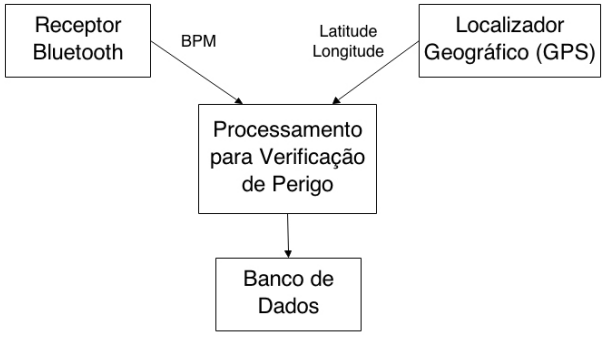

Figura 2.3: Fluxograma de Operação do MP

2.4 Situações de Anomalia

As situações que serão identificadas como anormais estão dispostas a seguir:

- **Situação 1**: Se a distância entre a PB e o MP for grande o suficiente para interromper sua comunicação;
- **Situação 2**: Se a PB for destruída ela cessará sua comunicação com o MP (análogo à Situação 1);
- **Situação 3**: Se o banco de dados cessar seu recebimento de informações sem o usuário ter desligado o sistema (por exemplo, destruição do MP);
- **Situação 4**: Se uma situação de perigo for identificada pelo aumento dos batimentos cardíacos;
- **Situação 6**: Se uma situação de perigo for identificada pela diminuição dos batimentos cardíacos. Neste caso um alarme sonoro local também será acionado, pois isso pode indicar situação de sonolência.

Ressalta-se ainda que:

- Optou-se por soar um alarme na diminuição da frequência cardíaca pois foi pensado na possibilidade do usuário dormir enquanto dirige, desta maneira o sistema protege o usuário também nestas situações;
- Ao ligar o sistema, o usuário deverá colocar seu CPF e senha, isso evitará que pessoas utilizem indevidamente o sistema;
- O sistema contará com uma interface simples e enxuta, de modo a não reter a atenção do usuário;
- O intuito da central de segurança será o de possuir um servidor que ficará verificando o banco de dados por uma *flag* de anomalia.

Capítulo 3

Embasamento Teórico

Como já explanado anteriormente, este trabalho possui grande multidisciplinaridade e abrange diversas áreas da engenharia e de programação. Desta maneira, serão apresentados os conteúdos teóricos que formaram o embasamento dos alunos para o desenvolvimento do trabalho.

Devido ao fato de alguns embasamentos teóricos possuírem conteúdo bastante extenso, optou-se pela criação de seus respectivos apêndices, aumentando assim a organização, compreensão e fluidez de leitura do trabalho.

3.1 Fisiologia do Estresse

O estresse pode ser resumido como um processo de respostas do organismo aos estímulos nocivos. E que, como resultado, altera o metabolismo como um todo. Estresse está relacionado com a ansiedade, ou seja, com o desejo de finalizar a sensação física e psíquica do contato com a realidade, acarretando assim em medo.

Por ser um assunto muito vasto e de grande complexidade, desenvolveu-se o Apêndice A para um melhor detalhamento do estudo realizado.

3.2 Dispositivos Móveis

Os dispositivos portáteis como *smartphones* e *tablets* vem ganhando mercado graças à sua crescente capacidade de processamento e de armazenamento, bem como sua facilidade de uso, leveza e portabilidade. Somado a essas características, a grande maioria dos aparelhos possuem câmeras, sensores de distância e luz, acelerômetro, GPS, comandos de voz e telas

sensíveis ao toque o que permite manipulação sem mouse e teclado.

Com tantas funcionalidades, os mesmos estão abrindo uma nova gama de oportunidades para sua aplicabilidade como novas e alternativas ferramentas de pesquisa e trabalho. O Apêndice B aborda algumas características de maneira mais aprofundada sobre tais aparelhos.

3.3 Programação Orientada a Objetos

Com o surgimento da Engenharia de *Software* várias técnicas de programação foram desenvolvidas visando aumentar a qualidade e produtividade dos *softwares*. Desta maneira, verificou-se que a reutilização de códigos era o segredo para promover a melhoria buscada pela Engenharia de *Software*. Desta maneira, foi verificado que as técnicas de programação estruturada não eram suficientes para alcançar satisfatoriamente o nível de melhoria buscado, assim sendo, na década de 60, surgiu a teoria de orientação a objetos [8].

De uma forma abstrata, o intuito básico da POO (Programação Orientada a Objetos) é o de aproximar o mundo real do virtual por meio do uso de Objetos. Assim sendo, o programador deverá moldar tais objetos e definir como deverá ser feita a interação entre os mesmos, de forma que atinjam funcionalidades requeridas para um correto funcionamento do *software*.

Em suma, a ideia principal de programação orientada a objetos consiste na possibilidade de aglomerar, em uma única estrutura de dados (classe), seus atributos (características) e seus métodos (funções). Desta forma, uma variável do tipo desta classe é chamada objeto, e este objeto conterá seus próprios dados e funções. As funções que um objeto pode executar são chamadas de métodos. Tais métodos, de modo geral, são o único meio de acesso aos seus campos de dados, chamados de variáveis de instância.

No Apêndice C serão explicadas as definições específicas da POO, bem como as mesmas serão caracterizadas de maneira mais profunda e clara.

3.4 Android

O Android é um sistema operacional móvel que roda sobre o núcleo Linux. Foi inicialmente desenvolvido pela Google e posteriormente pela *Open Handset Alliance*, sendo que a Google é a responsável pela gerência do produto e engenharia de processos. O Android permite aos desenvolvedores escreverem *software* na linguagem de programação Java controlando

o dispositivo via bibliotecas desenvolvidas pela Google [9]. Maiores detalhes sobre este assunto poderá ser verificado no Apêndice D.

3.5 UML

A UML, *Unified Modeling Language*, é um padrão internacional, que permite a especificação, construção e documentação de *softwares* que utilizem programação orientada a Objetos, desta maneira, a UML independe da linguagem de programação utilizada.

No Apêndice E deste trabalho serão descritos e exibidos os conceitos básicos, sintaxes e significados de alguns diagramas para que, futuramente, os mesmos possam ser utilizados para facilitar o entendimento e documentação, principalmente das classes, do *software* final.

3.6 Computação Na Nuvem

O conceito de computação na nuvem consiste, em suma, na utilização de recursos físicos remotos presentes na rede de computadores através da internet, permitindo, desta maneira, que sejam executadas diversas tarefas, como o de armazenamento de dados, independente do *hardware* utilizado, local físico e sem a preocupação com a estrutura física de *hardware*.

Um melhor detalhamento sobre o conceito, bem como sua classificação e pontos positivos e negativos serão descritos no Apêndice F.

3.7 Banco de Dados

Por definição, um banco de dados é uma coleção de dados logicamente coerente que possui um significado implícito cuja interpretação é dada por uma determinada aplicação [10].

Um BD é uma entidade na qual é possível armazenar dados de maneira estruturada e com a menor redundância possível, com o intuito de alimentar programas e usuários diferentes. Desta maneira, a noção básica de dados está acoplada geralmente a uma rede, a fim de poder compor um acervo de informações, daí o nome banco [11].

Vale ressaltar que dados e informações não são palavras sinônimas. Enquanto dados são fatos brutos, em sua forma primária, que muitas vezes não fazem sentido sozinhos, as informações consistem no agrupamento de dados de forma organizada para fazer sentido e gerar conhecimento.

Para maiores detalhamentos sobre os BDs foi elaborado o Anexo G, que contém informações adicionais sobre estes.

Capítulo 4

Métodos

O sistema proposto pode ser facilmente encarado como a interação entre dois módulos distintos, a PB e o MP, conforme apresentado no capítulo "Especificação do Projeto". Assim sendo, nesta seção serão apresentados os conceitos e métodos empregados para a elaboração das aplicações descritas anteriormente, detalhando também os materiais utilizados e *hardwares*/*softwares* desenvolvidos.

Para um melhor entendimento, as primeiras seções abordam o que foi desenvolvido para a PB, seguido pelo entorno do MP.

A figura 4.1 serve apenas como um parâmetro para facilitar o alinhamento entre as seções desenvolvidas neste capítulo com as partes do sistema proposto.

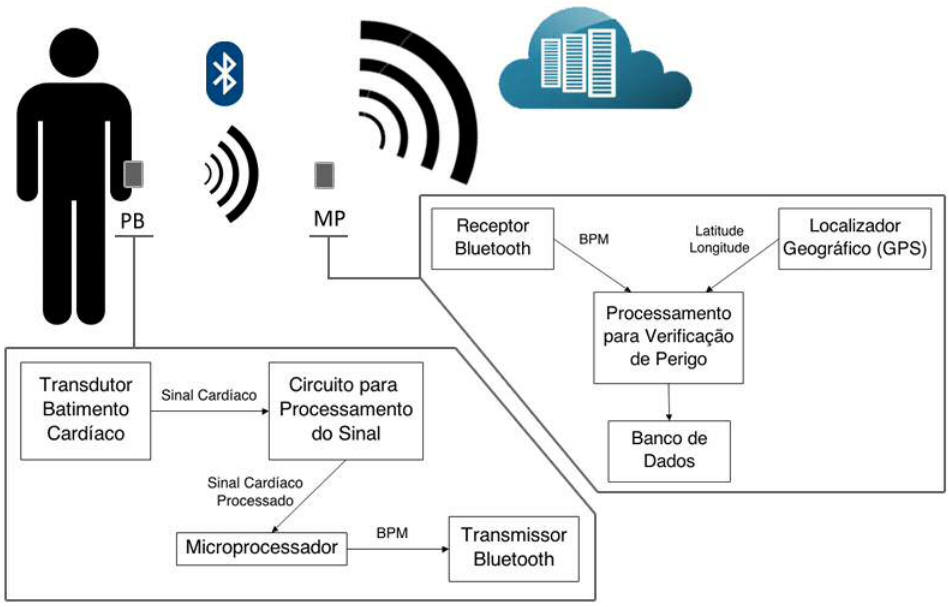

Figura 4.1: Esquema de Funcionamento do Sistema Completo

4.1 Coleta do Batimento Cardíaco - Pulseira Biométrica

Como o sistema baseia-se na identificação do perigo via comportamento da pulsação cardíaca, a escolha do sensor para aquisição de tal batimento torna-se fundamental para o trabalho.

Após diversas pesquisas para definir o sensor mais apropriado, optou-se pelo "*Pulse Sensor Amped*", que pode ser visto na figura 4.2.

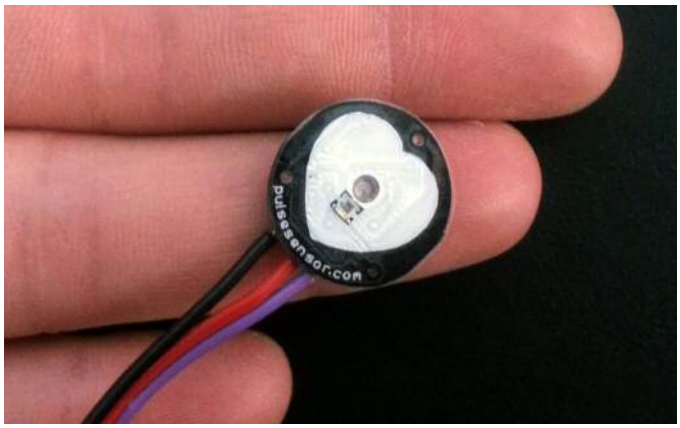

Figura 4.2: Transdutor *Pulse Sensor Amped*

4.1.1 Funcionamento do Sensor

Este sensor foi projetado com intuito inicial de ser utilizado na plataforma Arduino e ser de fácil aplicação. Possuindo apenas 3 pinos de conexão (alimentação, terra e saída do sinal), foi projetado para que o usuário simplesmente ligue o mesmo na alimentação, coloque-o em contato com, por exemplo, a ponta do dedo, e colete os dados referentes a resposta cardíaca [12].

Mesmo tendo sido projetado para Arduino, em termos funcionais, ele pode ser visto apenas como um sensor analógico, podendo assim ser utilizado em qualquer outra plataforma.

Seu funcionamento é relativamente simples: Na parte frontal do sensor (figura 4.2), pode ser verificado um pequeno buraco no meio, que é onde fica um LED na cor verde. Próximo a esse LED, pode-se verificar um pequeno quadrado, que corresponde a um sensor de luz ambiente (fotodiodo), exatamente como os encontrados em *smartphones* e *tablets* para ajustar automaticamente o brilho da tela e acordo com diferentes iluminações ambientes [12].

Desta maneira, quando o sensor é ligado, o LED fica constantemente aceso, iluminando a ponta do dedo ou o lóbulo da orelha (ou qualquer tecido que contenha vasos capilares em

seu entorno), enquanto o sensor de luz recebe a luz refletida. Tal luz refletida corresponde a refletividade relativa do sangue nos tecidos capilares, uma vez que a hemoglobina anexada a moléculas de oxigênio reflete uma quantidade de luz diferente da hemoglobina não anexada. Assim sendo, a resposta do sensor é análoga ao de uma pletismografia, ou seja, ela consiste na variação do volume dos vasos capilares, sendo tal variação identificada via diferentes concentrações de hemoglobina anexada com moléculas de oxigênio [13].

4.1.2 Operação do Sensor

Em teoria, de acordo com o fabricante, o sensor opera na faixa de 3 a 5V, contudo, após sua caracterização em laboratório, optou-se por trabalhar com o mesmo o mais próximo possível de 3v, pois é com uma alimentação nesse entorno que se consegue uma melhor resposta, como pode ser verificado na figura 4.3. Quando o sensor está ligado, porém não está em contato com tecidos capilares, sua resposta corresponde a um sinal DC de $v/2$ volts, sendo v a tensão de alimentação. O Apêndice H mostra os resultados obtidos da caracterização do mesmo.

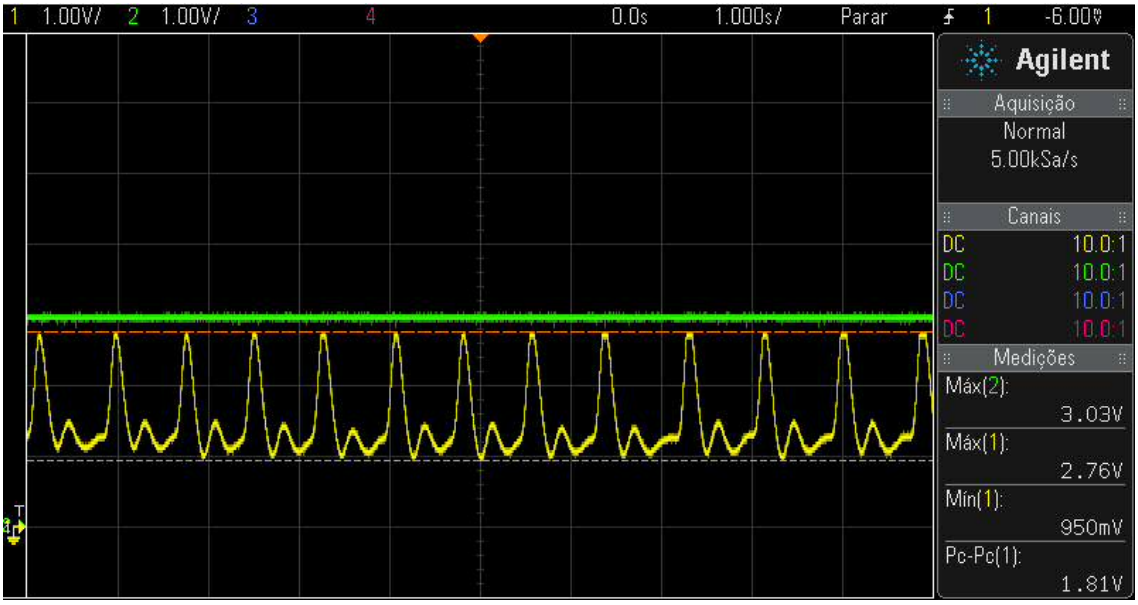

Figura 4.3: Resposta do Sensor Operando a 3v

4.2 Processamento do Sinal do Sensor - Pulseira Biométrica

Como já apresentado, o sensor possui uma resposta periódica (se o batimento cardíaco for constante) e contínua no tempo. Antes de enviar tal sinal para o microprocessador contar os

batimentos, foi necessário desenvolver um estágio intermediário para tratar tal sinal, de modo que o microprocessador pudesse interpretá-lo como um sinal digital.

Como neste projeto o intuito é contar os batimentos, foi proposto desenvolver um circuito para identificar a ocorrência do batimento, dando como resposta um sinal lógico dependendo da entrada provida pelo sensor cardíaco.

Como a resposta do sensor cardíaco é sempre a mesma, alterando-se apenas sua frequência (e, dependendo do usuário, alterando um pouco a amplitude do sinal), pensou-se na utilização de um circuito comparador de tensão.

4.2.1 Arquitetura do Circuito

O circuito comparador de tensão foi a arquitetura escolhida para processar o sinal do transdutor. Seu funcionamento consiste em comparar o sinal de entrada vindo do sensor cardíaco (V_{sensor}), com um sinal de tensão de referência. Sempre que o sinal de entrada for maior que o de referência, o comparador de tensão altera seu nível de tensão de saída, e quando o sinal de entrada volta a ser menor que o de referência, o comparador de tensão volta a ter como saída seu valor de tensão original.

A arquitetura do circuito pode ser vista na figura 4.4.

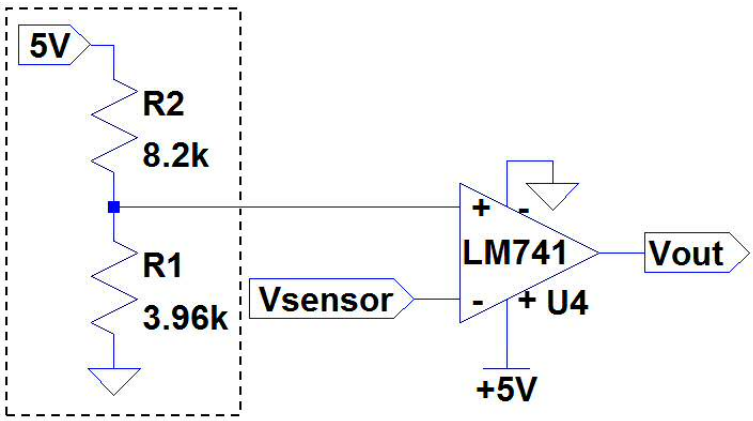

Figura 4.4: Arquitetura do Circuito Comparador de Tensão

A parte destacada figura 4.4 é a responsável por gerar a tensão de referência. Ela consiste num simples divisor resistivo, que segue o seguinte equacionamento:

$$V_{referencia} = \frac{R_1}{R_1 + R_2}.V_{in}$$

Onde V_{in} corresponde à tensão de entrada, no caso, 5v e $V_{referencia}$ a tensão de referência.

Contudo, para definir-se o valor de $V_{referencia}$, deve-se atentar para algumas variáveis:

- Sabe-se que o sensor, quando não está em contato com tecidos capilares, possui uma saída de $v/2$, onde v é a tensão de alimentação do sensor. Como definiu-se que o sensor seria alimentado com 3v, em vazio, o mesmo ficaria com um sinal constante de $1,5$ volts. Assim sendo, $V_{referencia}$ deve ser suficientemente maior que $1,5$ v;
- Quando em operação, o sensor possui uma forma de onda como mostrada na figura 4.5. Nesta figura está destacado o período da onda, no qual pode-se notar a existência dos picos referentes a sístole e a diástole (contração e relaxação dos ventrículos, respectivamente). Desta maneira:
  - O valor de $V_{referencia}$ deve ser menor que o valor de amplitude do primeiro pico (sístole), ou seja, 2,76v;
  - O valor de $V_{referencia}$ deve ser maior que o valor de amplitude do segundo pico (diástole), ou seja, 1,5v.

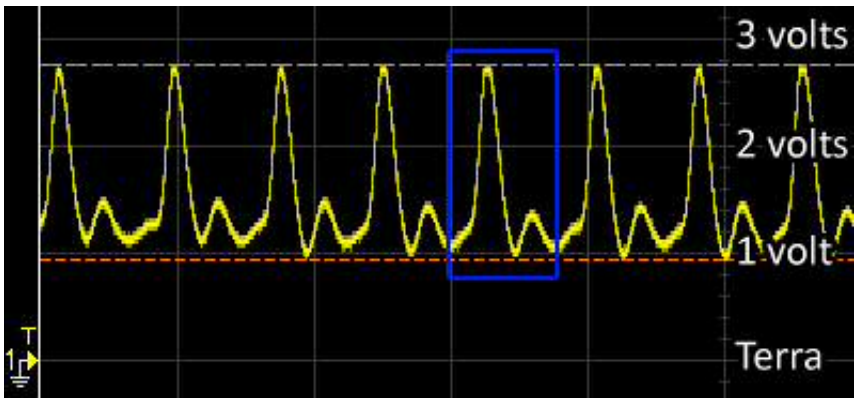

Figura 4.5: Detalhe da Forma de Onda Originada pelo *Pulse Sensor*, quando em operação

Assim sendo, admiti-se um valor de $V_{referencia}$ de 1,6v e desenvolveu-se os cálculos:

$$V_{referencia} = \frac{R_1}{R_1+R_2}.V_{in} \quad \rightarrow \quad 1,6 = \frac{R_1}{R_1+R_2}.5 \quad \rightarrow \quad R_1 = 0,32.(R_1+R_2) \quad \rightarrow$$

$$\rightarrow \quad R_2 = 2,125.R_1$$

Assim sendo, optou-se por $R_2 = 8,2k\Omega$ e $R_1 = 3,9k\Omega$ (valores comerciais). Dessa maneira, medidos os valores de resistores selecionados($R_2 = 8,20k\Omega$ e $R_1 = 3,96k\Omega$), pôde-se verificar o valor real de $V_{referencia}$:

$$V_{referencia} = \frac{R_1}{R_1 + R_2}.V_{in} \quad \rightarrow \quad V_{referencia} = \frac{3,96k\Omega}{3,96k\Omega + 8,20k\Omega}.5 \quad \rightarrow \quad V_{referencia} = 1,63volts$$

Desta maneira, toda vez que o sinal do sensor for mais que 1,63v, o circuito comparador de tensão mudará seu estado. Para verificar tais condições, foi feito a simulação do circuito (figura 4.6), utilizando uma senoide com *offset* DC de 1,5v e amplitude 1v.

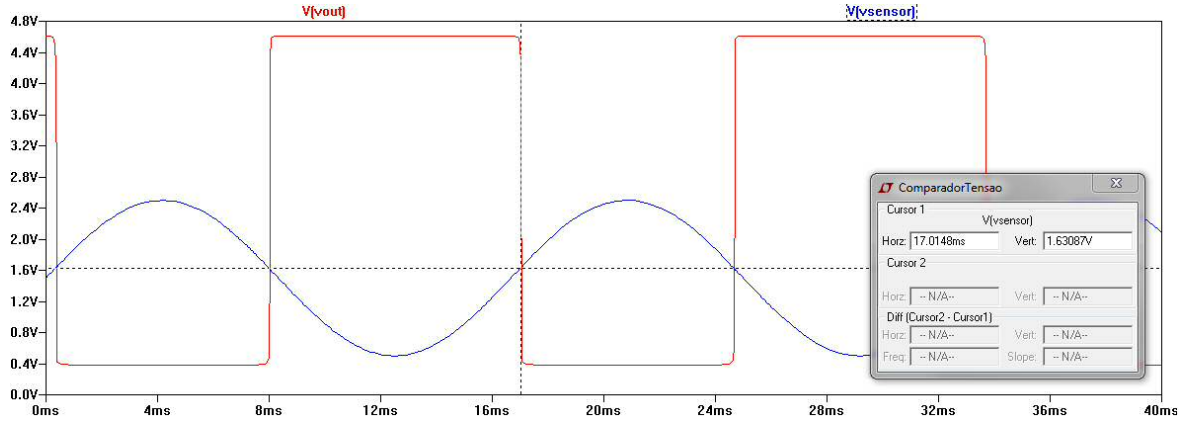

Figura 4.6: Resultado da Simulação do Circuito Comparador de Tensão

Após verificar o funcionamento do circuito na ferramenta de simulação, o mesmo foi montado e testado, obtendo o resultado da figura 4.7.

Figura 4.7: Comportamento do Circuito Comparador de Tensão Após Montado

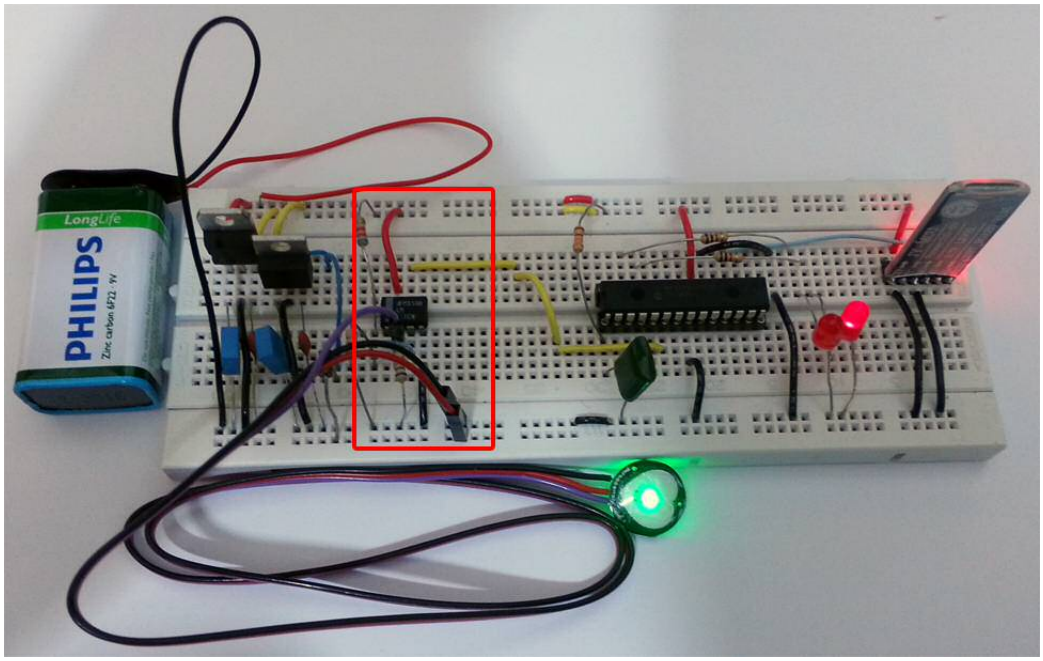

Figura 4.8: Circuito Montado (Destaque em vermelho para a parte do Compardor de Tensão)

4.3 Microprocessador - Pulseira Biométrica

Com o sinal vindo do sensor e tratado pelo circuito comparador de tensão, o microprocessador é então responsável por contabilizar os batimentos cardíacos e enviá-los ao módulo *bluetooth* via porta serial.

Para isto, foi escolhido o microprocessador 18F2550 da Microchip (figura4.9), que é um DIP28 (*Dual Inline Package* 28), ou seja, possui duas filas de 14 pinos, totalizando 28 (figura 4.10). Sua escolha se deu baseado em diversos fatores:

- Cada pino não tem uma única função. Dependendo das necessidades de cada aplicação, pode-se utilizar cada uma das funções de certo grupo de pinos;
- Possui 4 *timers* sendo 1 de 8 bits e o restante de 16 bits, 24 pinos de I/O, conversor AD de 10 bits;
- Tecnologia *nanoWatt*: Utilizando oscilação interna pode-se reduzir consumo de energia em até 90%; o *core* da CPU pode ser desabilitado enquanto seus periféricos são mantidos ativos, reduzindo seu consumo em até 4% do original; e seus modos de economia de energia podem ser chamados na própria programação, permitindo a otimização de consumo via *software*;

- Baixo custo (em torno de 17 reais) dado seu ótimo desempenho.

Assim sendo, ao mesmo tempo que esse PIC supre as necessidades do projeto, ele ainda possui capacidade o suficiente para futuras novas implementações no sistema, além de possuir baixo consumo quando oscilando internamente, característica essa essencial para a PB, que é um dispositivo móvel e necessita da maior autonomia possível.

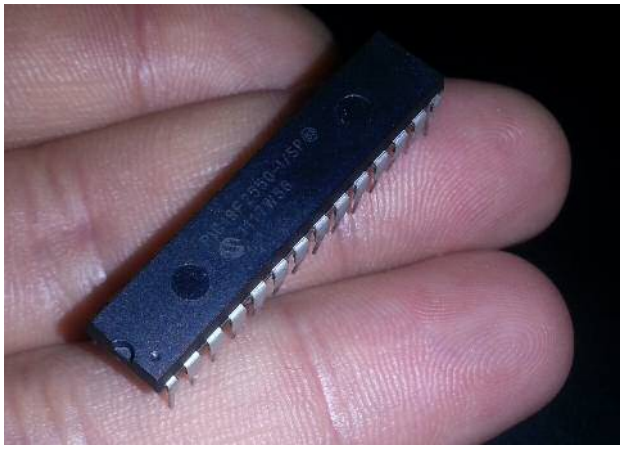

Figura 4.9: Microprocessador PIC 18F2550 da Microchip

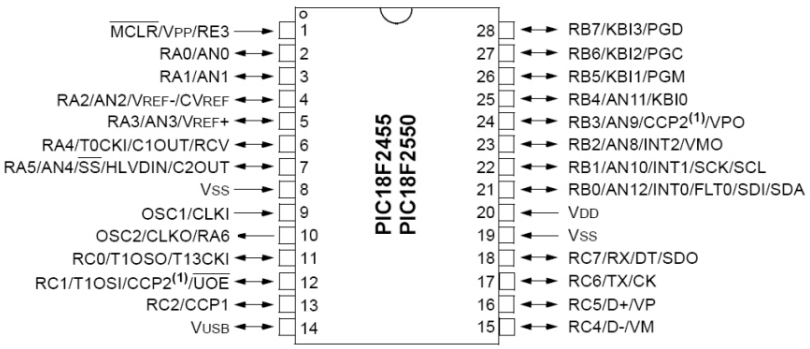

Figura 4.10: Diagrama de Pinos do Microprocessador PIC 18F2550

4.3.1 Arquitetura do Circuito

Como o microprocessador possui oscilador interno, a arquitetura do circuito de entorno do microprocessador fica bastante simples, não necessitando de capacitores e cristais para gerar a oscilação. A figura 4.11 mostra o circuito contendo dois LEDs para ajustes na programação e teste de controle de fluxo do programa, uma chave para *reset*, bem como o pino de entrada para o sinal do sensor já passado pelo comparador de tensão, identificado por SENSOR e o pino de envio de informações para o módulo *bluetooth*, identificado por DATA_OUT.

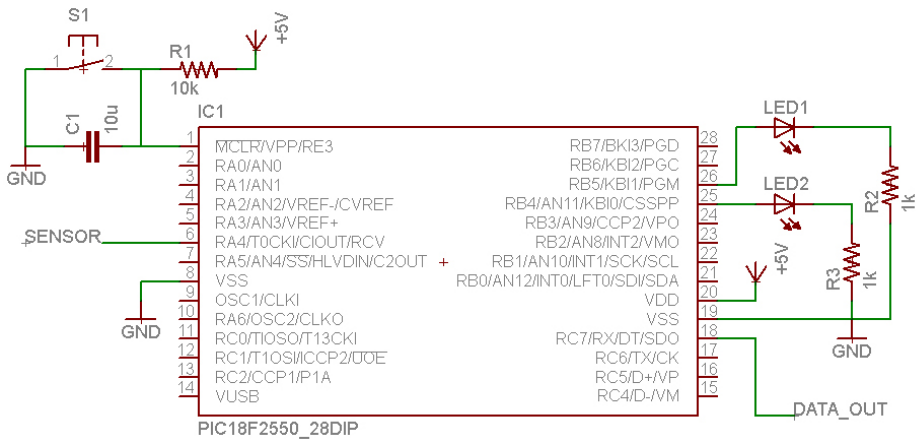

Figura 4.11: Arquitetura do Circuito de entorno ao Microprocessador

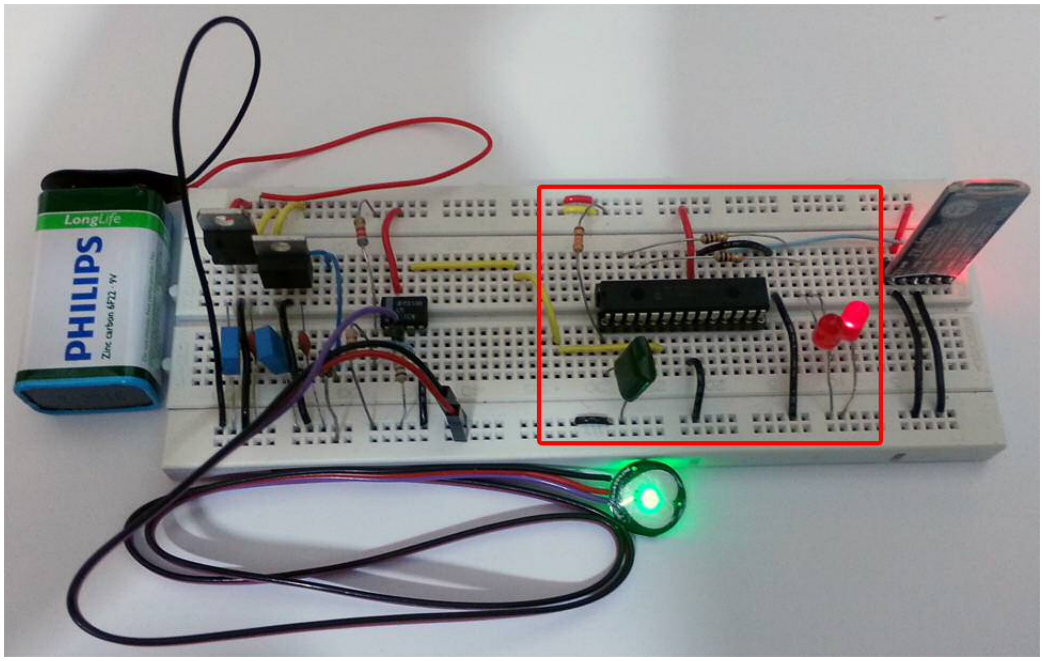

Figura 4.12: Circuito Montado (Destaque em vermelho para a parte do Microprocessador)

4.3.2 Programa Desenvolvido

O sinal provindo do comparador de tensão é binário, ou seja, ele vale ou 0v ou 5v. Além disso, a ocorrência de um batimento cardíaco pode ser identificada como uma descida de borda neste sinal, ou seja, para cada batida completa do coração, haverá uma descida de borda do sinal.

Baseado nisso, o programa desenvolvido utiliza o *counter 0* do microprocessador para contabilizar a quantidade de descidas de borda do sinal, numa janela de 15 segundos que é gerada a partir de um *delay* no programa. Ao final deste *delay*, multiplica-se o valor do

contador por 4, simulando assim uma janela de aquisição de 1 minuto e tal valor é enviado ao módulo *bluetooth*.

O envio do valor do batimento cardíaco segue o padrão *CDU (asterisco, centena, dezena e unidade), sendo o asterisco um caractere de sincronismo. Assim, um envio completo de um batimento consiste no envio separado do caractere de sincronismo, seguido pela centena do valor do batimento, dezena e por fim, unidade.

O código desenvolvido pode ser verificado no Apêndice I. Ressalta-se que este está completamente comentado, facilitando assim sua interpretação e provendo um maior detalhamento de seu fluxo e ações.

4.4 Módulo *Bluetooth* - Pulseira Biométrica

O módulo *bluetooth* utilizado foi o JY-MCU da fabricante Linvor, mais conhecido como HC-05 (nome de seu *firmware*). De maneira simplificada, sua operação consiste na conversão de sinais recebidos via interface serial para interface *bluetooth*. Apesar de ser um módulo bastante versátil e fácil de usar, sua documentação é um tanto quanto falha (contém alguns erros) e escassa.

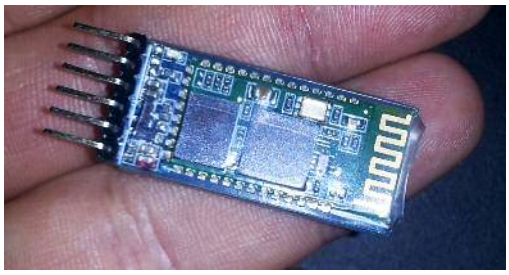

Figura 4.13: Módulo Bluetooth HC-05

4.4.1 Configuração

Mediante o uso dos chamados "comandos AT", deve-se configurar o módulo antes de iniciar sua utilização. Tais comandos permitem definir o nome no qual ele poderá ser descoberto, sua senha, baud *rate*, entre outros.

Para configurá-lo, deve-se seguir alguns simples passos em sua inicialização:

- Conecte os pinos GND, TXD e RXD, deixando os pinos VCC e KEY desconectados (figura 4.14);

- Coloque o pino KEY em nível alto. Este pino é o único que opera em 3.3v, todos os outros operam em 5v;
- Conecte o pino VCC;

Feito isso o módulo entrará no modo de configuração. Neste modo o módulo fica aguardando o recebimento de comandos AT via serial a um *baud rate* de 38400. A tabela 4.1 apresenta alguns dos comandos mais utilizados e importantes, sendo que os comandos podem ser verificados facilmente no *datasheet* do componente [14].

Função	Comando	Resposta	Parâmetro
Reset Dispositivo	AT+RESET	OK	Não há
Config. Comunicação	AT+UART=Param1,Param2,Param3	OK	Param1: baud rate; Param2: Se 0 -> 1 stop bit , se 1 -> 2 stop bits; Param3: bit paridade
Config. Nome	AT+NAME=Param	OK	Param: Nome desejado
Config. Senha	AT+PSWD=Param	OK	Param: Senha desejada

Tabela 4.1: Comandos AT Mais Utilizados e Importantes

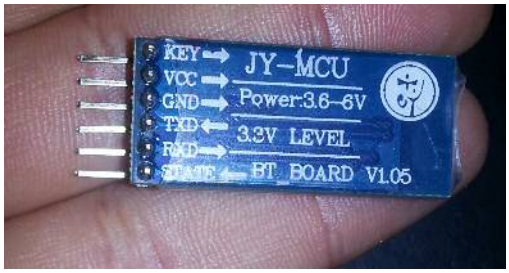

Figura 4.14: Identificação dos Pinos do Módulo Bluetooth HC-05

4.4.2 Programa Para Configuração

Utilizando os comandos AT necessários, o módulo foi configurado de acordo com o desejado. O programa utilizado para a programação pode ser verificado no Apêndice J. O código foi executado sobre o microprocessador PIC 18F2550, o mesmo descrito nas seções anteriores.

4.5 Descrição das Classes - Módulo de Processamento

Para detalhar o funcionamento do MP, optou-se por primeiro descrever as classes criadas, em seguida o fluxo do programa, para então apresentar o código. Como o Android é uma linguagem orientada a eventos e a objetos, a apresentação proposta torna seu entendimento mais claro.

4.5.1 MainActivity

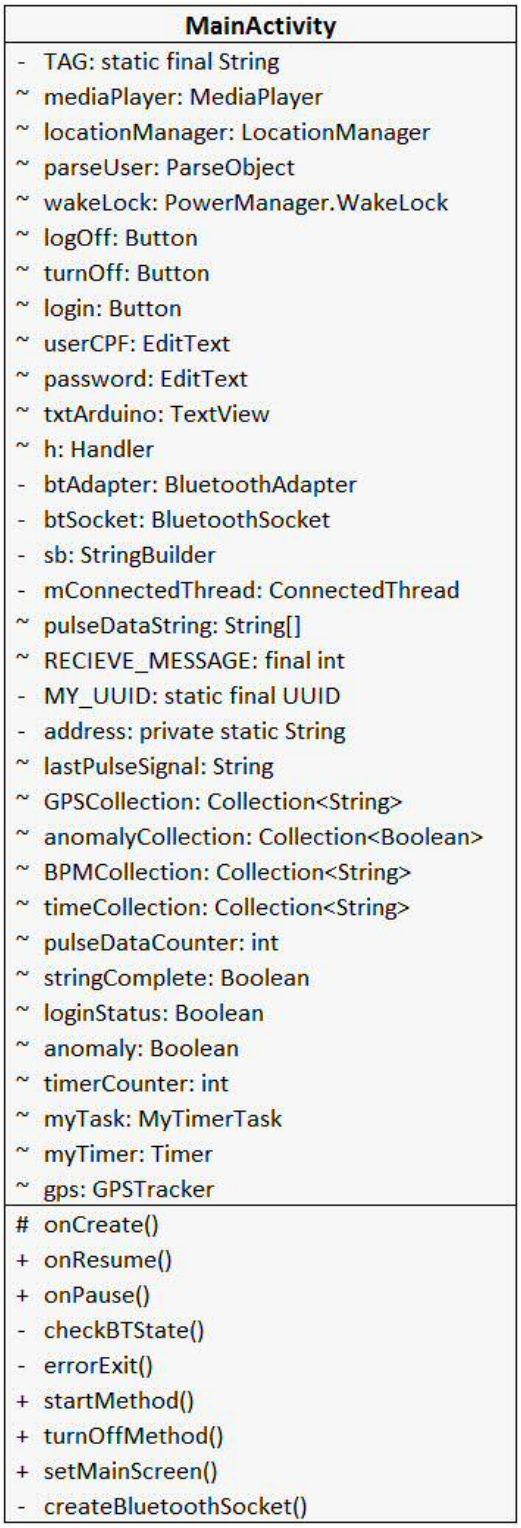

Figura 4.15: Diagrama de Classe UML da Classe MainActivity

Essa é a classe principal da aplicação inteira e é também a única classe que herda as características de uma atividade. Basicamente ao ligar a app, ocorre a configuração da interface com

o usuário e então é estabelecido um canal de comunicação com o dispositivo *bluetooth*.

O canal de comunicação supracitado trata-se de um *socket* especial para receber e enviar dados *bluetooth*. Neste caso, como haverá apenas a recepção de dados, criou-se apenas um canal para recepção. Para garantir que a UI *thread* não fique presa lendo os dados que chegam, cria-se uma *thread* que fará isso totalmente em paralelo.

4.5.2 ConnectedThread

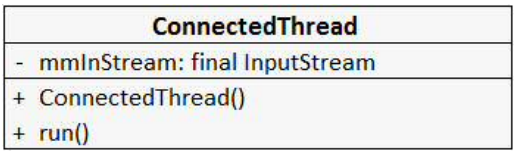

Figura 4.16: Diagrama de Classe UML da Classe ConnectedThread

Essa classe faz a leitura dos dados que entram através do canal de comunicação estabelecido entre o dispositivo móvel Android e o módulo *bluetooth*. Os dados recebidos são então armazenados num *buffer*. Esses dados são então enviados a um *Handler* cujo papel é manter um registro das últimas 10 leituras de batimentos cardíacos e fazer o processamento para verificar a ocorrência de anomalia.

Caso anomalia seja identificada e o usuário esteja logado, será emitido um alerta sonoro. Caso ele não estiver logado, nada acontecerá.

4.5.3 MyTimerTask

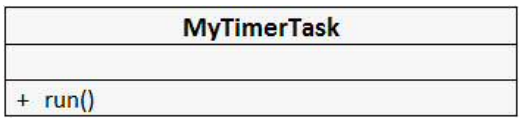

Figura 4.17: Diagrama de Classe UML da Classe MyTimerTask

Esta classe é usada para criar interrupções periódicas para que os últimos registros referentes à localização GPS, ocorrência de anomalia, batimento cardíaco e tempo sejam armazenados no banco de dados.

4.5.4 GPSTracker

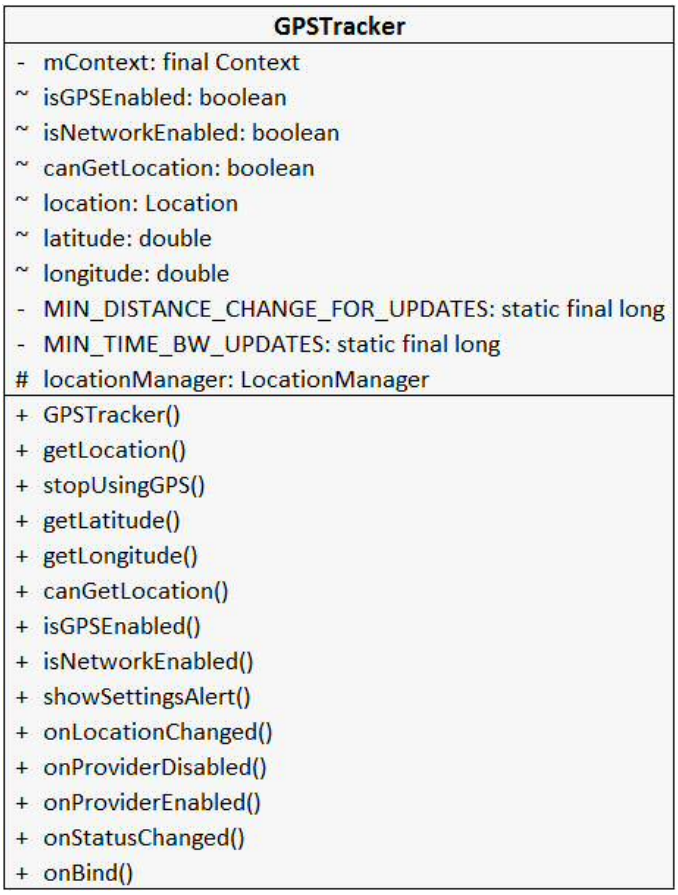

Figura 4.18: Diagrama de Classe UML da Classe GPSTracker

Classe construída com o único propósito de tornar modular a obtenção de informações sobre latitude e longitude do dispositivo móvel. Toda complicação é "escondida"numa única classe.

4.6 Fluxo do Programa - Módulo de Processamento

Aplicações para dispositivos Android não necessariamente seguem um único fluxo sequencial. Aliás, pelo contrário, são em sua maioria regidas por interrupções, eventos ou *threads* paralelas.

Tendo em vista esse panorama, são descritos os principais fluxos da aplicação individualmente, para que o todo possa ser compreendido pelo entendimento de cada parte individualmente. A figura 4.19 provê um fluxograma de apoio para o entendimento do fluxo de operação.

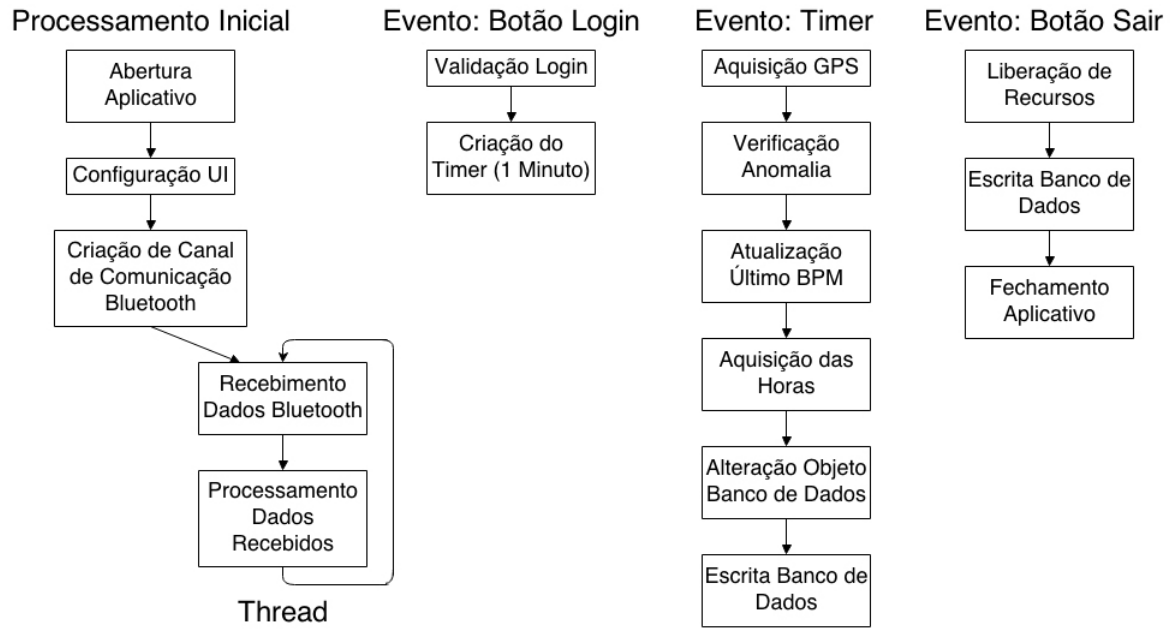

Figura 4.19: Fluxo do Programa Desenvolvido

4.6.1 Processamento Inicial

O processamento inicial refere-se à *thread* principal, ou UI *thread*. Ao iniciar a aplicação todo conteúdo gráfico necessário para interação com o usuário é inicializado. Logo em seguida, para que a aplicação já possa iniciar o recebimento de dados *bluetooth*, cria-se um canal de comunicação entre o celular e o módulo *bluetooth*. Dispara-se então uma *thread* que será responsável por ler os dados vindouros do canal e tratá-los.

Essa thread será executada toda vez que o smartphone receber dados via bluetooth. Conforme especificado anteriormente, os dados são enviados caractere a caractere, seguindo o formato *CDU (asterisco, centena, dezena e unidade), desta maneira, após lido os quatro caracteres referentes a informação sobre um batimento, esta informação é inserida no vetor dinâmico de 10 posições, que sempre contém os 10 últimos registros de batimentos cardíacos adicionados.

Ainda nesta *thread* ocorre a determinação de situação de risco devido o aumento ou diminuição dos batimentos cardíacos. Pelas informações contidas no vetor dinâmico determina-se a média dos batimentos cardíacos. Esta media é utilizada para o cálculo do valor máximo que o último batimento cardíaco recebido pode possuir. Para isto, utiliza-se a fórmula:

$$BPM_{Max} = 0.00384.media^2 - 0.357.media + 128$$

Se o último valor recebido superar tal valor máximo, levanta-se a *flag* anomalia que, posteriormente é escrita no banco.

Além disso, se o último batimento for menor que 50, novamente levanta-se a *flag*.

Para determinar tais valores limiares, partiu-se do fato que, em sonolência, o batimento pode chegar a 40 BPM, bem como, em estado acelerado,pode atingir até mesmo 200 BPM. Contudo, a variação porcentual frente a uma situação de stress é muito maior se o usuário estiver em repouso do que se estiver se exercitando, surgindo assim a necessidade da fórmula de limite máximo ser quadrática.

Por último, nesta *thread* também é coletado o horário do sistema, para que, no evento timer esse horário seja comparado com o horário da escrita, verificando se houve destruimento da PB (identificado pelo não recebimento de dados via bluetooth).

4.6.2 Evento: Botão Login

Após o usuário ter digitado o CPF e senha, e ter apertado o botão de *login*, uma busca é feita pelos objetos armazenados no banco de dados, e verifica-se a existência do objeto especificado. Caso o *login* seja efetuado com sucesso, o *timer* das interrupções periódicas é inicializado, bem como é indicado no banco que o usuário fez *login*.

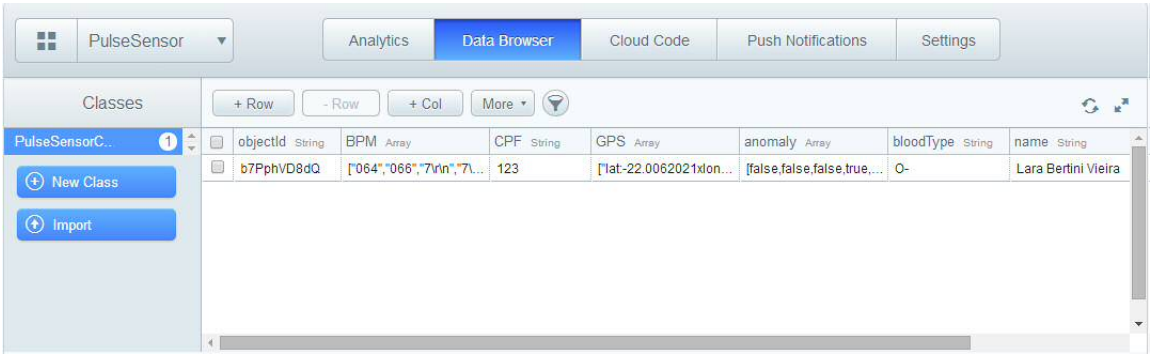

Figura 4.20: Interface do Banco de Dados Utilizado

4.6.3 Evento: Timer

Quando uma interrupção periódica ocorre, dados GPS são adquiridos e salvos no banco de dados, assim como dados referentes à ocorrência de anomalia, último registro de batimento cardíaco, e horário em que as informações estão sendo salvas no banco. Aqui também é verificado o horário do último recebimento de um dado via bluetooth, de modo que, se tal horário for grande, é enviado ao banco que ocorreu uma anomalia.

4.6.4 Evento: Botão Sair

Após o usuário ter apertado o botão de encerramento do aplicativo, os recursos alocados programaticamente são liberados, ocorre uma última escrita em banco e a aplicação é finalmente encerrada.

A última escrita no banco serve para identificar que o usuário fez *logoff* do sistema, isso serve para que um servidor externo possa verificar se MP foi destruído, ou seja, se usuário constar como logado e não tiver enviando informações, ocorreu uma anomalia.

4.7 Código do Programa - Módulo de Processamento

O código resultante do desenvolvimento da aplicação está presente no Apêndice K. As classes descritas acima, bem como o fluxo de execução podem ser verificados, sendo necessário apenas uma base inicial de programação orientada a objetos e java. Ressalta-se que o código foi completamente comentado a fim de proporcionar um maior entendimento sobre o mesmo.

As figuras 4.22 e 4.23 exibem a interface do programa desenvolvido, sendo executado em um *smartphone* modelo Galaxy S3, da fabricante Samsung 4.21. O código XML referente a interface também está presente no Apêndice K.

Figura 4.21: Smartphone Samsung Galaxy S3

Figura 4.22: Tela de login do Sistema

Figura 4.23: Tela de Execução do Sistema (aparece após usuário realizar *login*)

Capítulo 5

Resultados e Discussões

Durante o desenvolvimento da aplicação e principalmente do algoritmo de detecção de anomalias detectou-se a falta de dados para um ajuste preciso do programa. Tendo em vista este desafio, desenvolveu-se então um algoritmo de média dinâmica. Porém existem vários outros algoritmos possíveis, mais inteligentes, que podem se adaptar dinamicamente ao usuário e à situação em que o mesmo se insere porém, para uso destes, é necessário um conhecimento de um conjunto de dados relativos ao comportamento cardíaco frente ao medo inesperado. Contudo, apesar dessa falta de dados, o sistema pôde ser simulado.

As figuras 5.1 e 5.2 apresentam dois dos testes de simulação realizados. A primeira delas consiste no envio de valores pré-definidos de batimentos cardíacos de modo a verificar se o sistema identifica a anomalia referente ao aumento da frequência cardíaca que ocorreu. Nessa mesma imagem ainda estão presentes o valor da média e do limiar de disparo do alarme em relação ao dados enviados anteriores ao dado anômalo. A segunda delas consiste também em valores pré-definidos de batimentos cardíacos porém eles decaem, de modo a simular uma dormência do usuário.

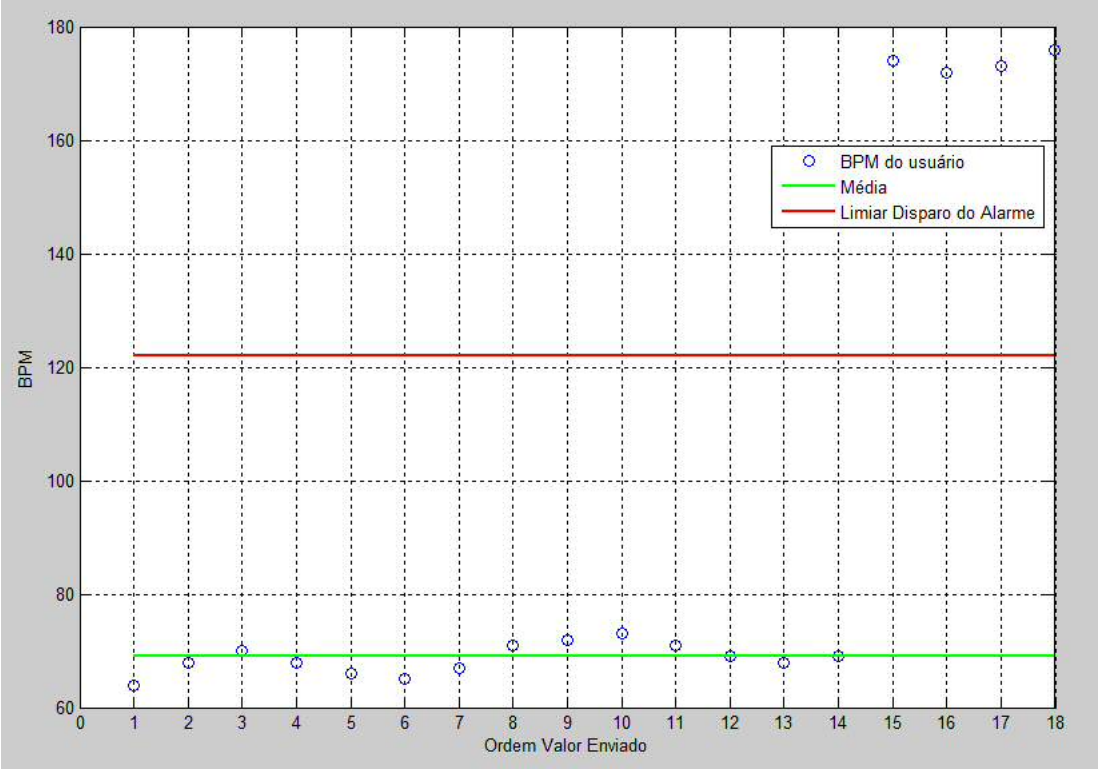

Figura 5.1: Exemplo de simulação para Disparo do Alarme referente Aumento da Frequência Cardíaca

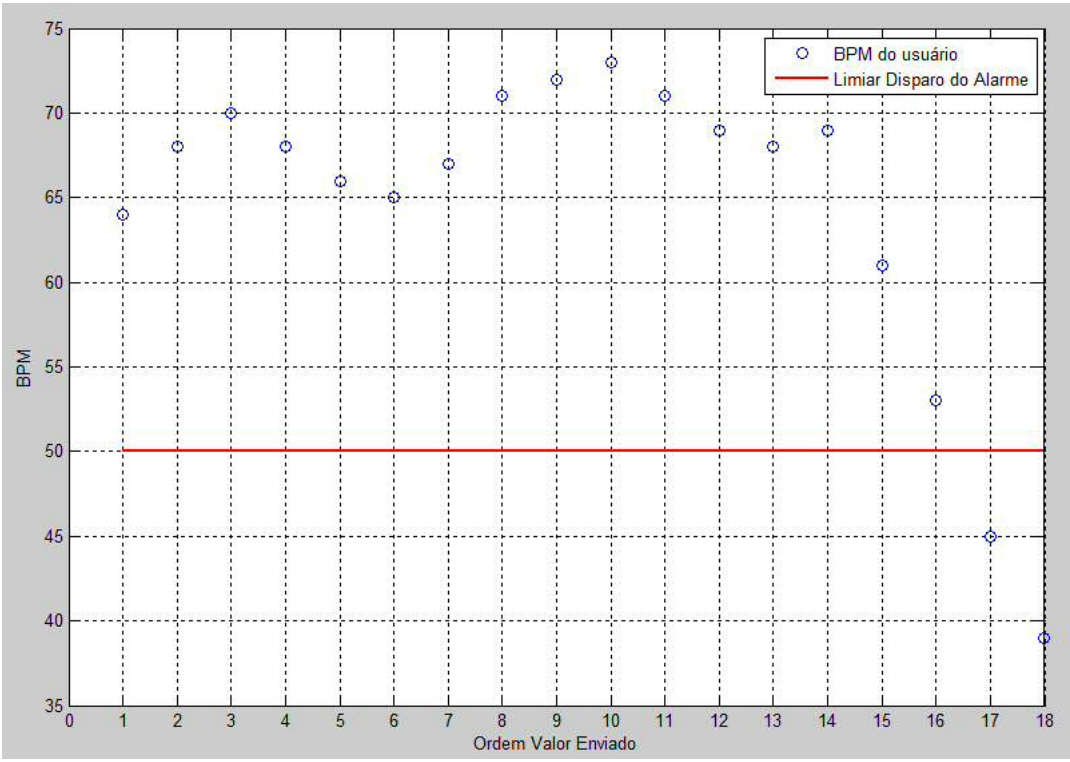

Figura 5.2: Exemplo de simulação para Disparo do Alarme referente Diminuição da Frequência Cardíaca

Ambos casos de teste mostraram a eficácia e precisão ao detectar as anomalias envolvidas.

Sabe-se que o medo pode ser identificado por intermédio do batimento cardíaco, mas a precisão e assertividade das anomalias detectadas aumentam com o aumento de tipos e quantidade de dados disponíveis para análise. Por exemplo, durante as pesquisas, verificou-se que a medida da impedância da pele, ou até mesmo a temperatura corporal do indivíduo são maneiras alternativas para identificação de situações extremas.

Apesar do sensor cardíaco ter demonstrado facilidade de manuseio, *hardware* encapsulado, custo baixo e ótima resposta para um indivíduo específico, foi notada a grande necessidade por maiores estudos a fim de encontrar a origem das diferentes respostas que o sensor demonstrou ao ser utilizado em indivíduos distintos. Maiores estudos são necessários para propor uma solução que possivelmente venha a funcionar como um esquema de calibração da reação do sistema à resposta do sensor. Outra alternativa estaria na utilização de um sensor mais robusto, cabe então uma avaliação do custo benefício da mudança. Em relação sua precisão, ela se mostrou bem próxima a de aparelhos comerciais existentes no mercado, como pode ser visto na tabela 5.1, que compara valores de pulsação medidos de três maneira distintas: pelo sistema desenvolvido, manualmente, e por um monitor automático de pressão arterial de pulso modelo HEM-631INT da fabricante Omrom *Healthcare*.

Sistema Desenvolvido	Manual	Monitor HEM-631INT Omrom *Healthcare*
70	73	73
71	73	74
73	74	74
96	93	92
112	116	114
116	116	114

Tabela 5.1: Comparativo do valor de BPM determinado por diferentes maneiras

Frente aos dados presentes na tabela, pode ser constatado que, apesar de o resultado não ser idêntico, são muito próximos, não influenciando assim no desempenho do sistema desenvolvido. Ressalta-se que os dados da tabla foram obtidos realizando as três medidas simultaneamente, em situação de repouso e exercício. As discrepâncias podem ser interpretadas pela aproximação de uma janela de 1 minuto a partir de valores coletados durante 15 segundos.

Para processar o sinal do sensor a escolha do comparador de tensão fui muito boa, por ser um circuito relativamente simples, não trouxe altos custos para o projeto. Alternativas mais rebuscadas como, por exemplo, um circuito do tipo comparador por histerese não traria

uma melhor detecção do pulso no caso abordado neste trabalho, trazendo assim apenas maior custo e complexidade para o sistema.

O desenvolvimento da parte de *software* do projeto dependia da escolha de um banco de dados na nuvem para armazenar um histórico dos dados do usuário. Cogitou-se a possibilidade de desenvolvimento de uma *App Engine App* que conversaria com um banco de dados não relacional (*Datastore*) da Google utilizando o Google *Cloud Endpoints*. Mas essa solução mostrou-se bem mais complexa que o esperado. Optou-se então por uma solução alternativa bem mais simples: Parse.

Parse é uma *startup* que oferece um sistema de gerenciamento de dados na nuvem. Eles oferecem um tipo de banco de dados orientado a objetos, permitindo que o desenvolvedor sincronize, escreva e leia informações da nuvem sem que seja necessário escrever uma única linha de código servidor (*back-end*) [15].

O BD Parse é *schemaless*, ou seja, faz uma abordagem NoSQL (não relacional) para armazenamento de dados [16] e permite que qualquer objeto seja salvo usando JSON. Funciona da seguinte maneira: cria-se uma aplicação e para essa aplicação criam-se classes e adicionam-se objetos a essas classes[15].

O Parse mostrou-se uma plataforma simples e eficiente. Além de oferecer um grande suporte técnico por meio de documentações completas e organizadas, a plataforma online para acesso às informações armazenadas na nuvem é direta e possui vários recursos importantes.

Como o Parse tem suporte à várias plataformas diferentes (iOS, Android, Windows Phone, Windows, entre outros). Portanto, o mesmo banco de dados pode ser usado em diferentes dispositivos, sejam eles móveis ou não. Lembrando que todas as outras plataformas possuem a simplicidade de integração via API em Android durante a fase de desenvolvimento deste trabalho.

Durante toda a fase de desenvolvimento e testes, o banco de dados funcionou corretamente e não houve problemas de conexão impeditivos para a leitura e escrita no banco. Constatou-se, porém, que quando da indisponibilidade de rede Wi-fi, o dispositivo utilizava rede 3G, e o tempo de acesso ao banco era longo. Não houve nenhum *black spot* que impediu a leitura/gravação de dados, mas caso houvesse, e o usuário estivesse logado, a situação seria interpretada como anomalia pelo servidor.

Vale ressaltar que a tendência mercadológica é que esses *black spots* sejam cada vez menores. A banda, atualmente reservada o sinal de TV analógica, será utilizada para telefonia móvel a partir de 2016. Portanto a tendência é de melhoria na cobertua de redes móveis e

cada vez menos ocorrências de perda de sinal.

O Parse, por ser um banco de dados orientado a objetos, encaixou-se perfeitamente às necessidades de uma app para android, por exemplo, pois o trabalho fora desenvolvido utilizando-se java como linguagem de programação.

A utilização de banco de dados como serviço na nuvem justificou-se pela facilidade em não ter que comprar servidores (ou outro hardware necessário) e programá-los. Além de ser mais barato e muito menos complexo, a utilização desse serviço na nuvem já garante que a ferramenta poderá ser facilmente escalável.

De modo geral, considerando os custos do projeto, pode-se citar os principais componentes de *hardware*, conforme tabela 5.2. Ressalta-se que os dados da tabela são valores aproximados considerando a taxa de conversão de 2,23 reais por dólar [17].

Dispositivo	Custo Médio (em reais)
PIC 18F2550	16,00
Pulse Sensor	55,75 (25 dólares)
Módulo Bluetooth HC-05	30
Total	101,75

Tabela 5.2: Principais Custos do Projeto

Desse modo, o custo de aproximadamente 100 reais do rastreador desenvolvido neste projeto, representa um porcentual de 1 a 1.6% em relação ao de um rastreador comercializado, que varia de 6 a 10 mil reais [2].

Para este trabalho não foi considerado o custo do dispositivo móvel por vários motivos: pela grande variedade de preços, levando em conta os vários tipos de dispositivos móveis, como celulares e *tablets*, além da existências de diferentes modelos com diferentes capacidades e recursos. Além disso, é crescente o número de indivíduos que possuem um dispositivo móvel em mãos, e portanto, pode-se tomar como ponto de partida que o público-alvo já o possuirá.

Foi utilizado um Galaxy S3 durante toda a fase de desenvolvimento deste produto. O Samsung Galaxy S3 é considerado dispositivo de médio a alto desempenho, portanto a aplicação rodou todos os momentos sem dificuldades, mesmo sendo necessário um processamento razoavelmente alto. O desafio que pode surgir é com relação ao tempo de duração da bateria. Claramente, a evolução do desenvolvimento das baterias não vem acompanhando o desenvolvimento de celulares.

A utilização do dispositivo móvel Android foi essencial para o desenvolvimento rápido

do aplicativo. Classes prontas em Android e *hardware* embutido no celular tornaram descomplicado, em vários sentidos, o desenvolvimento da parte de comunicação com o módulo *bluetooth* e com o banco de dados na nuvem. Não pode-se ainda avaliar o custo de desenvolvimento do mesmo sistema numa plataforma iOS. Esta última é um pouco mais fechada com relação à utilização de recursos próprios do dispositivo, o que poderia vir a acarretar maiores custos com licença.

Em geral, o sistema final desenvolvido mostrou-se mais flexível do que imaginado. Existem muitas outras aplicações nas quais pode-se utilizar o módulo desenvolvido: detecção quando um motorista está prestes a dormir com o veículo em movimento, prever se uma pessoa terá um mal súbito, auxiliar na detecção rápida de acidentes de trabalho em ambientes fabris, obtenção de dados biométricos de atletas, entre outros.

Como o sistema final ainda não foi integrado numa placa de circuito impresso única, ainda não há noção do tamanho que ficaria. Talvez o tamanho possa ser uma dificuldade a ser superada antes de inserir o produto no mercado, pois a premissa é fazer com que o mesmo seja parecido com uma pulseira e pequeno o suficiente para ser usado como tal. As poucas pessoas contatadas demonstraram interesse no produto, mas a quantidade amostral conseguida certamente não é suficiente para afirmar que o produto poderia vir a ter uma boa receptividade no mercado. Outro fator que aumentaria também o tamanho da pulseira seria a inserção de uma bateria capaz de alimentar todo o módulo biométrico do sistema. No protótipo foram utilizados dois reguladores: um de 5 e outro de 3 volts a fim de garantir que a tensão ficasse num nível compatível aos outros componentes do módulo biométrico. Portanto, antes de encapsular o sistema, seria necessário efetuar uma leve mudança na arquitetura do circuito de modo a dispensar os reguladores e consequentemente diminuir o consumo de potência e espaço do circuito em geral.

Capítulo 6

Conclusão

A partir da proposta inicial, foi possível realizar a conclusão do projeto em sua totalidade. Vale ressaltar que anteriormente à ideia de utilização do dispositivo móvel Android como módulo de processamento, cogitou-se a possibilidade de utilização de um processador acoplado a um módulo GPRS para envio de dados para o banco de dados na nuvem e um módulo *bluetooth* para recebimento de dados vindouros da pulseira. As dificuldades encontradas com essa proposta eram muitas: desenvolvimento complexo, indisponibilidade de um módulo GPRS para os principais kits de processamento do mercado, e alto custo. Todos esses desafios, somado ao crescente número de pessoas que tem acesso a um *smartphone*, fez com que a abordagem mudasse, garantindo que o celular passasse a ser o módulo responsável pelo de maior trabalho de processamento do projeto. Além de reduzir o custo do projeto, todo *hardware* necessário para desenvolvimento do projeto já está embutido e integrado com um *software* de fácil acesso através de classes prontas. O custo do projeto também diminuiu consideravelmente, já que assume-se que o usuário tenha disponível um *smartphone*.

Saindo um pouco do escopo geral do projeto, vale a pena citar algumas observações de cada parte, individualmente.

O sensor de pulso, por exemplo, apresentou um sinal de resposta aparentemente dependente das características particulares da pele de um indivíduo. Foi implementado um comparador de tensão com um certo valor de referência para contabilizar o número de batimentos cardíacos por minuto. Porém, notou-se a necessidade de maiores estudos, pois há uma certa probabilidade do formato, frequência e amplitude da onda de resposta do sensor variar de pessoa para pessoa. Neste caso, haveria a necessidade de calibrar o equipamento para qualquer usuário diferente que queira se cadastrar no sistema. Caso a calibragem não ocorra, é muito provável que a contagem de pulsos por minuto não seja efetiva.

O microprocessador utilizado na PB possui ainda vários pinos disponíveis para trabalhar com mais sensores, podendo medir valores de temperatura, impedância e outros. A assertividade da análise de anomalia pode melhorar dependendo dos dados extras a que se tem acesso.

A comunicação *bluetooth* mostrou-se estável durante todo o desenvolvimento do sistema. Em nenhum momento ocorreu interferência ou perda de dados numa transmissão, além da vantagem da não utilização de fios para comunicação. Além disso, as classes e métodos prontos em Android para trabalhar especificamente com comunicação *bluetooth* facilitaram muito o desenvolvimento do *software* deste projeto.

O custo total deste projeto ficou bem abaixo dos dispositivos comerciais atuais do mercado. Muitas melhorias ainda devem ser feitas para que este sistema esteja apto à comercialização, mas o custo em si, não será um impeditivo.

Através dos estudos realizados sobre a dinâmica de batimentos cardíacos e a resposta humana ao medo, pode-se averiguar que o pulso é uma variável indispensável para identificação de situações extremas para o usuário, e portanto, pode ser utilizado para reconhecer um sentimento, algo abstrato, como o medo, por exemplo.

Os autores deste trabalho creem que a tendência mercadológica está no crescimento da utilização de sensores para obtenção de dados biométricos que possam ser utilizados para fins de segurança individual.

Finalmente, o conhecimento agregado a partir do desenvolvimento deste trabalho de conclusão de curso foi muito grande, principalmente pelo fato do projeto envolver todo ciclo de desenvolvimento de um dispositivo, iniciando-se pelo projeto inicial, passando pela construção do *hardware* e terminando com o desenvolvimento do *software*.

6.1 Trabalhos Futuros

Para trabalhos futuros sugere-se a utilização de mais sensores a fim de aumentar a assertividade das análises que possivelmente serão feitas pelo módulo de processamento. O processador da pulseira biométrica ainda possui vários pinos para sensores não utilizados e que poderiam ser alvos de melhoria. Adicionar mais variáveis ao projeto, como impedância de pele, temperatura, entre outras, poderia inclusive ser uma maneira interessante de obter dados para análise e estudos das respostas humana à diferentes situações.

Com um banco de dados mais completo em mãos, um outro foco de melhoria pode estar

no algoritmo de detecção de anomalias. Há vários algoritmos disponíveis, inclusive utilizando inteligência artificial, que poderiam entender e se adequar a cada usuário individualmente.

Sugere-se também a utilização do sistema em empresas: quando um funcionário tiver um mal súbito, o ambulatório imediatamente será avisado e a ajuda estará a caminho o mais rápido possível. O mesmo pode-se aplicar para familiares com problemas de saúde, como apneia, arritmia, sopro, entre outros.

Adicionar microfone ou microcâmera ao sistema também pode ser visto como uma boa opção para aumentar a visibilidade da conjuntura a que o usuário está inserido.

Desta maneira, nota-se que o sistema pode ser mais trabalhado e personalizado de acordo com a aplicação, porém mantendo duas de suas principais características: autonomia de identificação de anomalias e portabilidade.

Referências Bibliográficas

[1] Android Website. *Distribuição das Versões do Android*. http://developer.android.com/resources/dashboard/platform-versions.html, Acesso em: 25 de Maio de 2014.

[2] Revista Eletrônica Exame. *Os Melhores Acessórios Contra os Roubos de Carros*. http://exame.abril.com.br/seu-dinheiro/noticias/os-melhores-acessorios-contra-os-roubos-de-carros, Acesso em: 18 de setembro de 2013.

[3] ABESE Associação Brasileira das Empresas de Sistemas Eletrônicos de Segurança. *FAQ*. http://www.abese.org.br/faq.asp, Acesso em: 01 de Junho de 2013.

[4] CNI Confederação Nacional da Indústria e IBOPE. *Pesquisa CNI-IBOPE: Retratos da Sociedade Brasileira*. Confederação Nacional da Indústria. Brasília DF, Outubro 2011.

[5] De-Segurança. *Roubo de Cargas do Brasil*. http://www.de-seguranca.com.br/index.php/artigos/gestao-de-riscos/seguranca-empresarial/494-roubo-de-cargas-do-brasil-generalidades, Acesso em: 04 de Julho de 2013.

[6] CaféPoint. *Índice de Roubos de Cargas Preocupa Exportadores de Café no Cerrado Mineiro*. http://www.cafepoint.com.br/cadeia-produtiva/giro-de-noticias/indice-de-roubos-de-cargas-preocupa-exportadores-de-cafe-no-cerrado-mineiro-81370n.aspx, Acesso em: 30 de Junho de 2013.

[7] IBOPE Instituto Brasileiro de Opinião Pública e Estatística. *Em 23 Anos de Democracia, Brasileiros Mudam de Opinião Sobre os Principais Problemas do País*. http://www.ibope.com.br/pt-br/noticias/Paginas/Em-23-anos-de-democracia-brasileiros-mudam-de-opiniao-sobre-os-problemas-do-Pais.aspx, Acesso em: 28 de junho de 2013.

[8] A. Sintes. *Sams Teach Yourself Object Oriented Programming in 21 Days*. Sams, 2001.

[9] R. R. Lecheta. *Google Android: Aprenda a Criar Aplicações Para Dispositivos Móveis com o Android SDK*. Novatec, 2010.

[10] Edson Norberto Cáceres. *Banco de Dados - Conceitos Básicos*. Universidade Federal do Mato Grosso do Sul - http://www.dct.ufms.br/~edson/bd1/bd1.pdf, Acesso em: 26 de maio de 2014.

[11] Kioskea. *Banco de Dados - Introdução*. http://pt.kioskea.net/contents/65-bancos-de-dados-introducao, Acesso em: 20 de maio de 2014.

[12] Pulse Sensor. *Pulse Sensor Open Hardware*. http://pulsesensor.myshopify.com/pages/open-hardware, Acesso em: 10 de setembro de 2013.

[13] KickStarter. *Pulse Sensor: an Open Source Heart-rate Sensor that Rocks*. https://www.kickstarter.com/projects/1342192419/pulse-sensor-an-open-source-heart-rate-sensor-that, Acesso em: 14 de setembro de 2013.

[14] HC-05. *HC-05 Bluetooth to Serial Module Port*. http://www.electronicaestudio.com/docs/istd016A.pdf, Acesso em: 06 de abril de 2014.

[15] Code. *Getting Started With Parse*. http://code.tutsplus.com/tutorials/getting-started-with-parse--net-28000, Acesso em: 07 de maio de 2014.

[16] Info Queue Brasil. *PostgreSQL: Armazenamento de Dados em Formato "schemaless"*. http://www.infoq.com/br/articles/postgresql-schemaless, Acesso em: 02 de maio de 2014.

[17] UOL Economia. *Cotações*. http://economia.uol.com.br/cotacoes/, Acesso em: 01 de junho de 2014.

[18] Marilza Bertassoni Alves Mestre e Neuza Corassa. *Da Ansiedade a Fobia*. Revista Psicologia Argumento. ISSN 0103-7013. Ano XVIII - No XXVI. Pg 105 - 126, 2000.

[19] Centro de Psicologia Especializado em Medos. *Estudos Científicos*. http://www.medos.com.br/estudos-cientificos/, Acesso em: 03 de Outubro de 2013.

[20] Edna Paciência Vietta. *Ansiedade: Reação de Luta-ou-Fuga*. http://viettaed23.blog.terra.com.br/2013/04/28/ansiedade-reacao-de-luta-ou-fuga/, Acesso em: 15 de Outubro de 2013.

[21] Revista Super Interessante e Istituto de Psicologia da Universidade de São Paulo. *A história da Morte*. http://super.abril.com.br/historia/historia-morte-446779.shtml, Publicação Original em Dezembro de 2006. Acesso em: 15 de Outubro de 2013.

[22] *How Stuff Works* Como Tudo Funciona. *Como Funciona o Medo*. http://pessoas.hsw.uol.com.br/medo.htm, Acesso em: 22 de Setembro de 2013.

[23] Universidade Federal Fluminense. *Material de Apoio à disciplina Fisiologia VI: Sistema Nervoso Autônomo*. http://www.uff.br/fisio6/PDF/sistema_cardiovascular/sistema_nervoso_autonomo.pdf, Acesso em: 08 de Outubro de 2013.

[24] Association of the British Pharmaceutical Industry Resources for Schools. *Hormones and Their Effects*. http://www.abpischools.org.uk/page/modules/hormones/horm8.cfm, Acesso em: 02 de Outubro de 2013.

[25] Businessweek. *Before IPhone and Android Came Simon, the First Smartphone*. http://www.businessweek.com/articles/2012-06-29/before-iphone-and-android-came-simon-the-first-smartphone, Acesso em: 09 de dezembro de 2013.

[26] IBM Website. *O Que é Mundo Pós-PC*. https://www.ibm.com/developerworks/mydeveloperworks/blogs/ctaurion/entry/o_que_e_o_mundo_pos-pc?lang=en, Acesso em: 15 de outubro de 2013.

[27] Benedict Evans from Enders Analysis. *The Mobile Explosion*. Publisher's Forum 2013 - http://publishersforum.de/wp-content/uploads/2013/04/BenedictEvans_MobileExplosion.pdf, Acesso em: 22 de abril de 2014.

[28] Business Insider. *The Number Of Smartphones In Use Is About To Pass The Number Of PCs*. http://www.businessinsider.com/number-of-smartphones-tablets-pcs-2013-12, Acesso em: 30 de abril de 2014.

[29] Portal G1. *Tablet Ultrapassa Vendas de Desktop e Notebook Pela Primeira vez no Brasil*. http://g1.globo.com/tecnologia/noticia/2014/03/tablet-ultrapassa-vendas-de-desktop-e-notebook-e-pela-1-vez-no-brasil.html, Acesso em: 14 de maio de 2014.

[30] Veja. *Vendas de Tablets Vão Superar as de Notebooks no Brasil*. http://veja.abril.com.br/noticia/vida-digital/vendas-de-tablets-vao-superar-as-de-notebooks-no-brasil, Acesso em: 14 de maio de 2014.

[31] IBGE. *Projeção da População do Brasil Por Sexo e Idade - 1980 a 2050*. http://www.ibge.gov.br/home/estatistica/populacao/projecao_da_populacao/2008/projecao.pdf, Acesso em: 20 de maio de 2014.

[32] IDG Now. *Até 2017 o Brasil Terá 70,5 Milhões de Usuários de Smartphones em Uso*. http://idgnow.com.br/blog/circuito/2014/01/22/base-de-usuarios-de-smartphones-na-america-latina-vai-aumentar-283-em-2014/, Acesso em: 20 de maio de 2014.

[33] P. Coad e E. Yourdon. *Análise Baseada em Objetos*. Campus, 1992.

[34] G. Pollice e D. West. *Use a Cabeça! Análise e Projeto Orientado ao Objeto* B. McLaughlin. O'Reilly, 2007.

[35] L A. Silva. Apostila de Android Website. *Android: Programando Passo a Passo*. http://www.apostilaandroid.net/, Acesso em: 25 de Novembro de 2013.

[36] M. L. da Silva e L. C. O. Pereira. *Android Para Desenvolvedores*. Brasport, 2009.

[37] H. M. Deitel e P. J. Deitel. *Java - Como Programar*. Prentice Hall, 2005.

[38] Como Tudo Funciona. *Como Funciona a Computação nas Nuvens*. http://www.businessweek.com/technology/content/aug2005/tc20050817_0949_tc024.htm, Acesso em: 10 de outubro de 2013.

[39] Open Handset Alliance. *Members*. http://www.openhandsetalliance.com/oha_members.html, Acesso em: 15 de novembro de 2013.

[40] Open Handset Alliance. *Overview*. http://www.openhandsetalliance.com/oha_overview.html - Tradução por Enzo Bertini Vieira e Lara Bertini Vieira, Acesso em: 20 de novembro de 2013.

[41] Jornal O Estado de São Paulo. *A era dos PCs está chegando ao fim*. http://economia.estadao.com.br/noticias/negocios%20tecnologia,a-era-dos-pcs-esta-chegando-ao-fim,59238,0.htm, Acesso em: 02 de janeiro de 2014.

[42] Canalys. *Android Takes Almost 50% Share of Worldwide Smartphone Market*. http://www.canalys.com/newsroom/android-takes-almost-50-share-worldwide-smart-phone-market, Acesso em: 08 de janeiro de 2014.

[43] Latin Post. *Android Market Share in 2013 Dominates at 80 Percent but Slower Growth Could Mean Trouble*. http://www.latinpost.com/articles/6598/20140129/android-market-share-2013-dominates-80-percent-slower-growth-mean.htm, Acesso em: 15 de janeiro de 2014.

[44] Oracle Website. *Download Java Development Kit*. http://www.oracle.com/technetwork/java/javase/downloads/index.html, Acesso em: 21 dez. 2011.

[45] Android Website. *Download Plugin ADT*. http://developer.android.com/sdk/eclipse-adt.html, Acesso em: 15 de Novembro de 2013.

[46] Android Website. *Download Android SDK*. http://developer.android.com/sdk/index.html, Acesso em: 15 de Novembro de 2013.

[47] Eclipse Website. *Download Eclipse IDE*. http://www.eclipse.org/downloads/, Acesso em: 20 de Novembro de 2013.

[48] Android App Market. *Android Architecture - The Key Concepts of Android OS*. http://www.android-app-market.com/android-architecture.html, Acesso em: 20 de fevereiro de 2014.

[49] Rafael Gomes Jean Fernandes e Vinicius Ferreira. *Sistema Operacional Android*. Universidade Federal Fluminense - http://www.midiacom.uff.br/~natalia/2012-1-sisop/tgrupo1.pdf, Acesso em: 02 de fevereiro de 2014.

[50] Android On Board. *Meu Primeiro Aplicativo Android*. http://androiddevbr.wordpress.com/2013/08/13/meu-primeiro-aplicativo-android/, Acesso em: 25 de janeiro de 2014.

[51] Hachi Tecnologia. *Android - Criando Seu Primeiro Projeto*. http://blog.hachitecnologia.com.br/mobile/android-criando-seu-primeiro-projeto, Acesso em: 18 de fevereiro de 2014.

[52] Android Developers. *Processes and Threads*. http://developer.android.com/guide/components/processes-and-threads.html, Acesso em: 12 de abril de 2014.

[53] Carsten Vogt. *Events and the UI Thread in Android*. Universitat Politècnica de València - https://www.youtube.com/watch?v=fN3t5BmBOiE, Acesso em: 11 de abril de 2014.

[54] M. C. Sampaio e E. L. Neto. *Material Sobre UML*. Universidade Federal de Campina Grande Centro de Engenharia Elétrica e Informática. http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/uml/, Acesso em: 02 de Outubro de 2013.

[55] SISNEMA. *Cloud Computing - O Novo Modelo de Computação*. http://sisnema.com.br/Materias/idmat019433.htm, Acesso em: 02 de março de 2014.

[56] InfoEscola. *Computação em Nuvem*. http://www.infoescola.com/informatica/computacao-em-nuvem/, Acesso em: 02 de março de 2014.

[57] InfoWester. *O que é Cloud Computing (Computação Nas Nuvens)?* http://www.infowester.com/cloudcomputing.php, Acesso em: 04 de março de 2014.

[58] Oficina da Net. *O que é Computação Nas Nuvens (Cloud Computing)?* http://www.oficinadanet.com.br/artigo/923/computacao_nas_nuvens, Acesso em: 03 de março de 2014.

[59] Tecnologia & Gestão. *Cloud Computing*. https://tecnologiaegestao.wordpress.com/2010/06/28/cloud-computing/, Acesso em: 05 de março de 2014.

[60] Google. *Google Docs*. https://docs.google.com/, Acesso em: 07 de março de 2014.

[61] Microsoft. *Microsoft SharePoint Online*. http://office.microsoft.com/pt-br/sharepoint, Acesso em: 08 de março de 2014.

[62] Microsoft. *Microsoft Azure*. http://azure.microsoft.com/pt-br/, Acesso em: 07 de março de 2014.

[63] Amazon Web Services. *Amazon Elastic Computing Cloud (EC) 2*. http://aws.amazon.com/pt/ec2/, Acesso em: 05 de março de 2014.

[64] Koding. *Koding*. https://koding.com/Home, Acesso em: 06 de março de 2014.

[65] Microsoft. *Microsoft Office Lync*. http://office.microsoft.com/pt-br/lync/, Acesso em: 06 de março de 2014.

[66] Parse. *Parse Cloud*. https://parse.com/, Acesso em: 12 de dezembro de 2013.

[67] Fabrício Rodrigues Henriques da Silva. *Um Estudo Sobre os Benefícios e os Riscos de Segurança na Utilização de Cloud Computing*. Trabalho de Conclusão de Curso - UNISUAM - http://fabriciorhs.files.wordpress.com/2011/03/cloud_computing.pdf, Acesso em: 29 de abril de 2014.

[68] Como Tudo Funciona. *Como Funciona a Computação nas Nuvens*. http://tecnologia.hsw.uol.com.br/computacao-em-nuvem.htm, Acesso em: 07 de março de 2014.

[69] João Eduardo Ferreira. *Introdução a Banco de Dados*. Instituto de Matemática e Estatística - Universidade de São Paulo - http://www.ime.usp.br/~jef/apostila.pdf, Acesso em: 28 de abril de 2014.

[70] Fernando Albuquerque. *Banco de Dados*. Universidade de Brasília - http://www.cic.unb.br/~fernando/matdidatico/apostilas/resumo/bdadossp.pdf, Acesso em: 10 de maio de 2014.

Apêndice A

Fisiologia do Estresse

O estresse é um processo de respostas do organismo aos estímulos nocivos, alterando o metabolismo de órgãos (rins, baço, intestino, glândulas sudorípara e salivar) e reorganizando o organismo para essa reação aos estressores. Dessa maneira, as funções digestiva, renal e sexual ficam inibidas, enquanto os músculos, esqueleto, coração, pulmões, e sistema linfático ficam mobilizados [18] [19].

A ansiedade se traduz por pressa, ânsia por movimento, inquietação interior, aflição do corpo, para que aquilo que estiver acontecendo ou que se suspeita acontecer, seja logo concluído. Em outras palavras, a ansiedade é o desejo de acabar logo, de terminar, de cessar a sensação física e psíquica do contacto com a realidade, nessa perspectiva esse desejo acaba sendo o representante do instinto de morte. [20]

Uma pesquisa realizada com jovens e adolescentes entre 13 e 17 anos consistia no questionamento do que mais gerava medo nos entrevistados, deixando as respostas em aberto. Esse estudo chegou a conclusão de que os 10 maiores medos relatados foram de ataque de terroristas, aranhas, morte, fracassar na vida, guerra, alturas, violência, solidão, do desconhecido (futuro), e guerra nuclear. E, desse estudo, o que se percebe nitidamente é que a maioria deles se relaciona ao medo de perder a vida.

O medo da morte é inerente ao desenvolvimento humano. Aparece na infância, a partir das primeiras experiências de perda, e tem várias facetas: trata-se de um medo do desconhecido, somado ao medo da própria extinção, da ruptura da teia afetiva, da solidão e do sofrimento. [21].

A.1 O Medo

O medo consiste na resposta gerada pelo cérebro, decorrente de um estímulo de estresse (como, por exemplo, o ataque de um aracnídeo, assalto, acidente de carro ou grandes alturas), acarretando na liberação de compostos químicos que causam, entre outras reações, o aumento da frequência cardíaca e a aceleração na respiração (conhecido por reação de luta ou fuga).

O cérebro pode ser compreendido como uma enorme e complexa rede de comunicação composta por mais de 100 bilhões de células nervosas que regem os sentimentos, pensamentos e atitudes dos humanos. Parte dessas comunicações levam ao pensamento e à ação consciente, enquanto outras produzem respostas autônomas, sendo o medo uma resposta quase inteiramente autônoma, ou seja, é disparada automaticamente pelo cérebro, de modo que um humano não consegue, por vontade própria, dispará-la.

Apesar de existirem diversas áreas do cérebro responsáveis pelo sentimento de medo, algumas partes possuem atuação mais efetiva, como mostrada na figura A.1 [22].

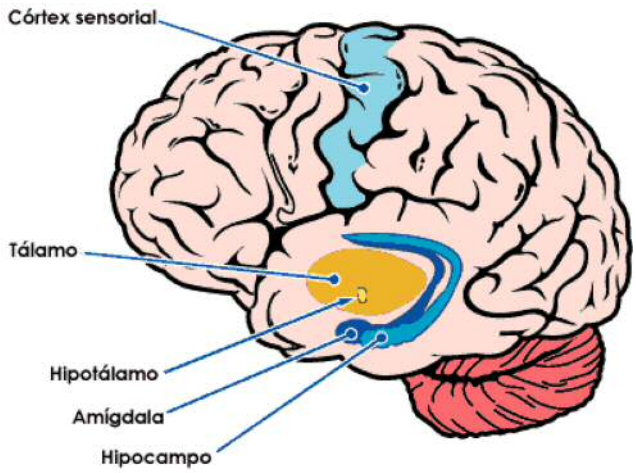

Figura A.1: Principais Partes do Cérebro Envolvidas na Reação de Medo

De maneira geral, as funções exercidas pelas regiões cerebrais da figura A.1, sao:

- **Tálamo**: Região cerebral que toma a decisão para onde enviar os dados sensoriais recebidos, sejam eles providos pela visão, audição, tato, olfato ou paladar;
- **Córtex Sensorial**: Responsável pela interpretação dos dados sensoriais;
- **Hipocampo**: Armazena e busca memórias conscientes, processando os estímulos de forma a estabelecer um contexto para os mesmos;

- **Amígdala**: Decodifica emoções, determinando possíveis ameaças e armazenando-as nas chamadas "memórias do medo"(conjunto de lembranças que estão correlacionadas a experiências de medo previamente enfrentadas);

- **Hipotálamo**: Responsável pela ativação da reação de "luta ou fuga".

A.2 O Processo de Criação do Medo

Como já apresentado, o processo de criação do medo acontece involuntariamente. Ele pode ser dividido em dois caminhos: O caminho baixo gera uma reação imediata e desordenada, enquanto o caminho alto gera uma reação não tão imediata, porém mais ponderada e precisa. Ressalta-se que ambos os processos ocorrem simultaneamente, conforme indicado pela figura A.2. As figuras A.3 e A.4 exemplificam o processo tomado nos caminhos baixo e alto, respectivamente.

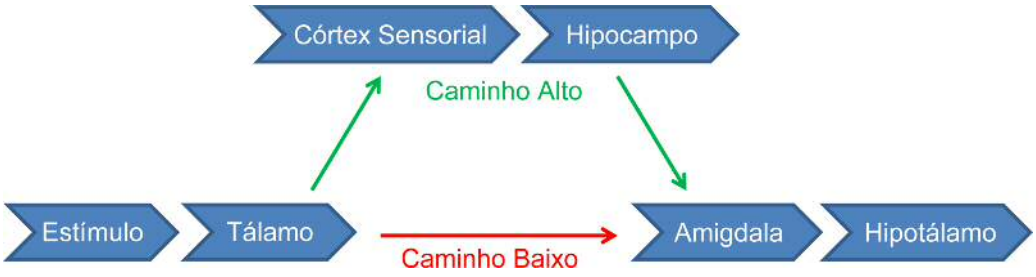

Figura A.2: Comparativo do Caminho Baixo e Caminho Alto

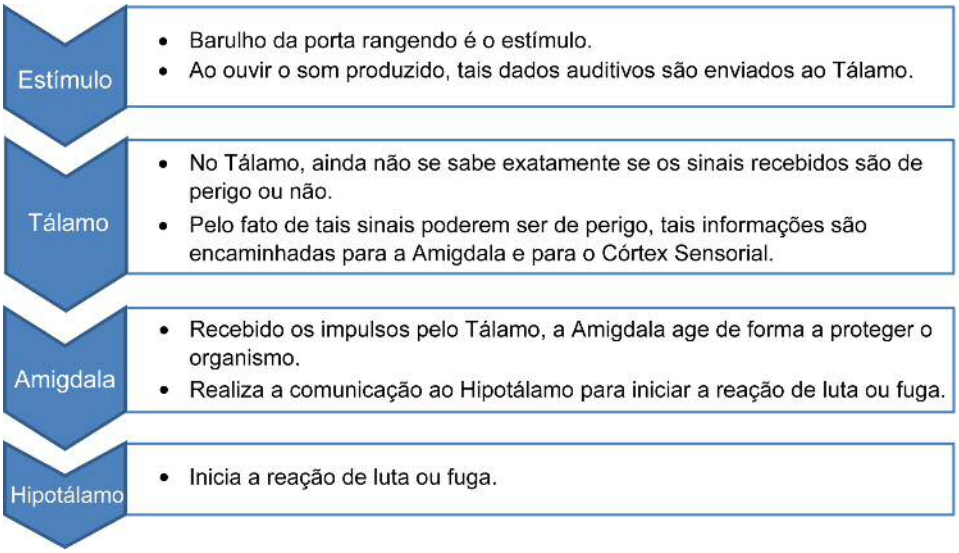

Figura A.3: Fluxograma de Exemplificação do Caminho Baixo

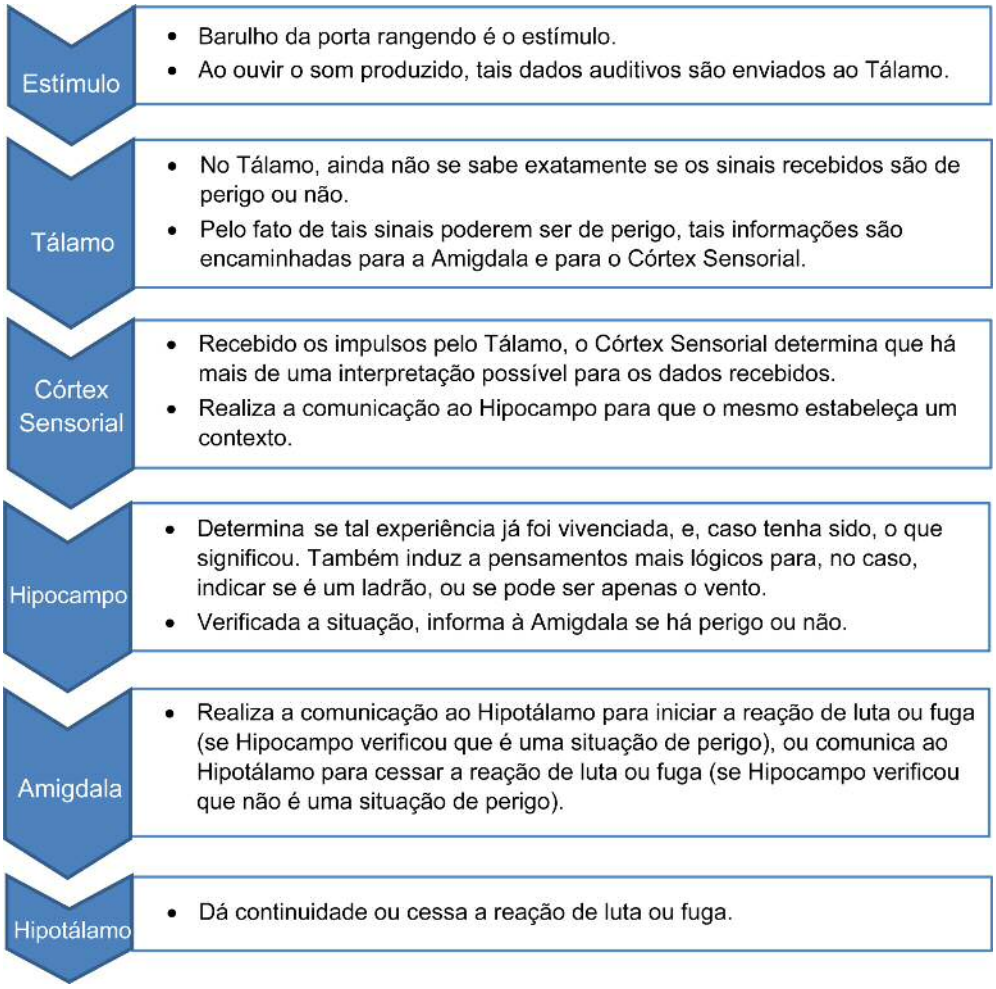

Figura A.4: Fluxograma de Exemplificação do Caminho Alto

A.3 Reação de Luta ou Fuga

Para produzir a reação de luta ou fuga, o hipotálamo ativa o sistema nervoso simpático e o sistema adrenocortical.

O sistema nervoso simpático utiliza as vias nervosas para iniciar reações no corpo, enquanto o sistema adrenocortical utiliza a corrente sanguínea. Desta maneira, é denominado "Reação de Luta ou Fuga"os efeitos combinados dos dois sistemas.

Quando o hipotálamo informa ao sistema nervoso simpático a necessidade de uma reação ao perigo, o efeito geral consiste na aceleração do corpo, ficando tenso e mais alerta. O sistema nervoso simpático envia impulsos para as glândulas e músculos lisos e informa à glândula adrenal para liberar adrenalina e noradrenalina na corrente sanguínea. A adrenalina e noradrenalina, por sua vez, são conhecidas por "hormônios do estresse"efetuam várias mudanças no corpo, incluindo um aumento na frequência cardíaca e na pressão sanguínea [22]

[23].

Simultaneamente, o hipotálamo livra o fator de liberação de corticotropina na glândula pituitária, ativando o sistema adrenocortical. A glândula pituitária secreta o hormônio ACTH (hormônio adrenocorticotrópico) que, através da corrente sanguínea, alcança o córtex adrenal, o qual ativa a liberação de aproximadamente trinta hormônios diferentes para preparar o corpo para lidar com uma ameaça [22].

A figura A.5 exibe o cascateamento de efeitos resultantes da ação do hipotálamo [22].

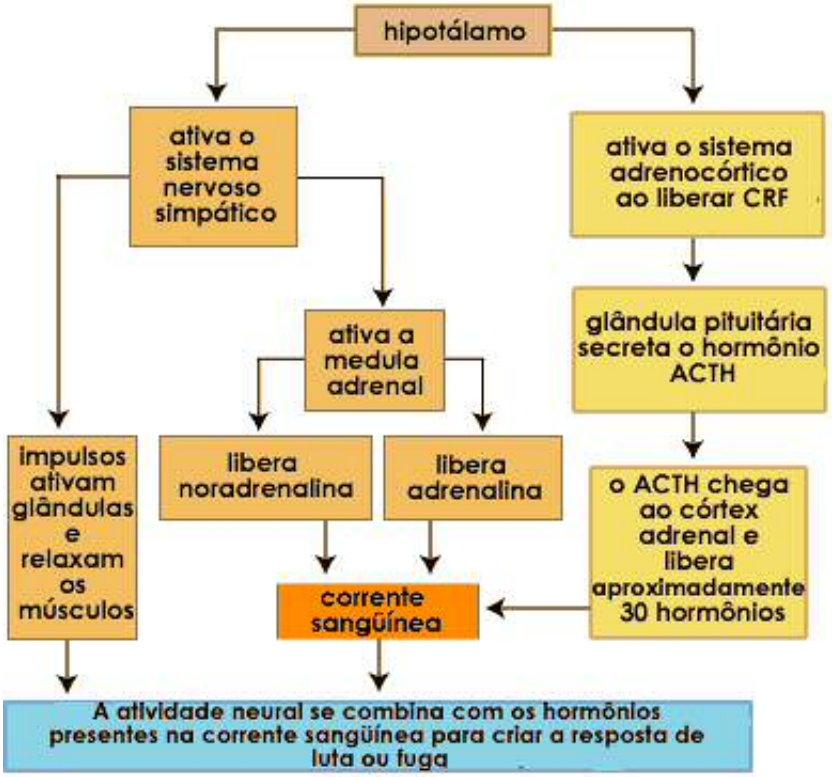

Figura A.5: Fluxograma de Ação do Hipotálamo

A.3.1 Efeitos da Reação de Luta ou Fuga

Dentre os efeitos presentes numa situação de luta ou fuga, ressalta-se [23]:

- Liberação de hormônios da glândula supra renal (especialmente a adrenalina, como será apresentado posteriormente);
- Inibição da motilidade (capacidade dos intestinos de realizarem movimentos peristálticos) e das secreções intestinais, aumentando a captação de nutrientes que antes seriam excretados e que agora serão essenciais para o fornecimento de energia;

- Ereção dos pêlos, em função dos músculos piloeretores;
- Constrição dos vasos esplâncnicos;
- Aumento da atividade cerebral, com foco apenas sobre a situação de risco;
- Aumento do metabolismo basal (quantidade de energia que o corpo necessita para funcionar);
- Aumento da atividade do músculo miocárdio (atividade cardíaca), vasodilatação das coronárias e vasoconstrição dos vasos sanguíneos periféricos, aumentando dessa forma a circulação e a pressão sanguínea, reduzindo em contrapartida as chances de hemorragias nas regiões mais periféricas, que podem ser afetadas pelo estímulo ameaçador;
- Aumento da atividade das glândulas sudoríparas, causando maior transpiração;
- Inibição dos músculos da bexiga urinária e excitação do esfíncter urinário, evitando assim a necessidade de micção;
- Sistema imunológico é inibido.

A.3.2 Adrenalina

É um hormônio produzido pelas glândulas supra-renais (adrenais) que se encontram na parte superior dos rins. Em condições normais, apenas uma pequena quantidade de adrenalina é produzida, o que ajuda a manter uma pressão sanguínea normal. Em situações de estresse e, consequentemente, de medo, na qual o corpo pode estar envolvido numa situação de luta ou fuga, uma grande quantidade de adrenalina é produzida muito rapidamente, o que acarreta numa série de efeitos drásticos no corpo [24] [23]:

- Elevação dos níveis de açúcar no sangue, estimulando o fígado a transformar o glicogênio em glicose, aumentando assim a energia disponível para as células do corpo;
- Liberação de gordura no Sangue do Tecido Adiposo;
- Aumento da frequência cardíaca;
- Aumento do fluxo sanguíneo para os músculos;
- Redução do fluxo sanguíneo para a pele e os intestinos, para enviar uma quantidade de sangue mais significativa aos grupos musculares maiores;

- Dilatação dos bronquíolos, aumentando a captação de oxigênio e melhorando a capacidade respiratória;
- Dilatação das pupilas, aumentando a captação de luz.

Todos esses efeitos são adicionados aos efeitos da reação de luta ou fuga, citados anteriormente.

O Fluxo da adrenalina em situações normais e de estresse pode ser verificado nas figuras A.6 e A.7, respectivamente.

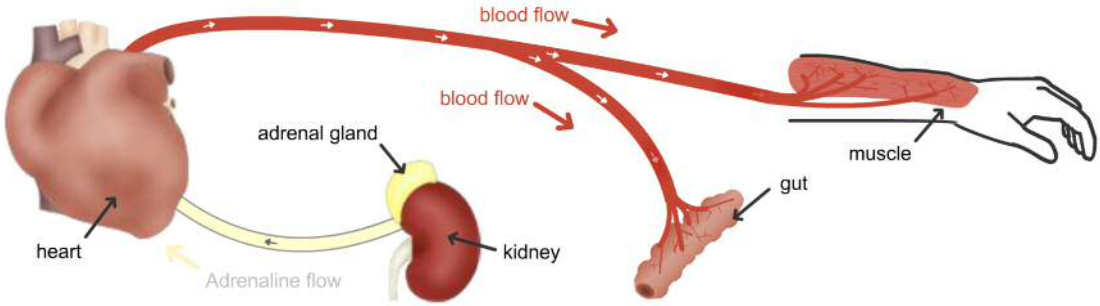

Figura A.6: Fluxo de Adrenalina em uma Situação Normal

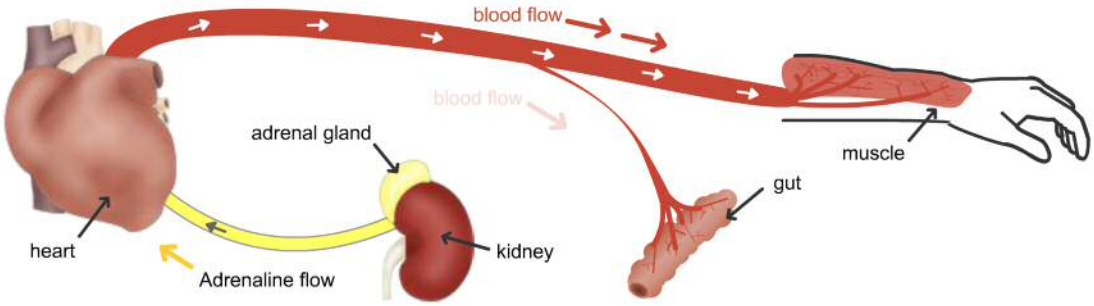

Figura A.7: Fluxo de Adrenalina em uma Situação Anormal de Estresse

Apêndice B

Dispositivos Móveis

B.1 Histórico e Evolução

O conceito de combinar a telefonia com a computação remete a meados de 1973, entretanto, apenas em 1993 a IBM conseguiu lançar um produto combinando as duas funcionalidades, o Simon. Desta maneira nasceu o primeiro *smartphone*. É interessante ressaltar que o termo *smartphone* apenas surgiu em 1997 pela empresa Ericsson [25].

A popularização dos *smartphones* veio no final da década de 1990 com os PDAs, através dos sistemas Palm OS e Windows CE, cabendo a BlackBerry dominar esse segmento de mercado até o lançamento do iPhone, pela Apple, em 2007. O iPhone veio para basicamente redefinir o conceito de *smartphones*, sendo o primeiro aparelho *multitouch* e sem teclado físico, e com capacidade de processamento muito mais elevada que qualquer antecessor. No ano seguinte e Google lançou o Android, sistema operacional gratuito, que é atualmente o mais usado nos *smartphones* [25].

B.2 Era pós-PC

O conceito de computador pessoal está sendo elevado a um novo patamar que consiste na intermitente presença de tais dispositivos ao longo do dia, uma vez que possuem baterias que lhes permite ficar horas ligados e até dias em *standby*. Esta realidade está sendo chamada de era pós-PC [26].

Um indicativo deste enorme crescimento que *smartphones* e *tablets* vem sofrendo pode ser notado na figura B.1 [27], que compara a quantidade de venda unitária trimestral (em milhões) de computadores pessoais, corporativos, *tablets* e *smartphones*.

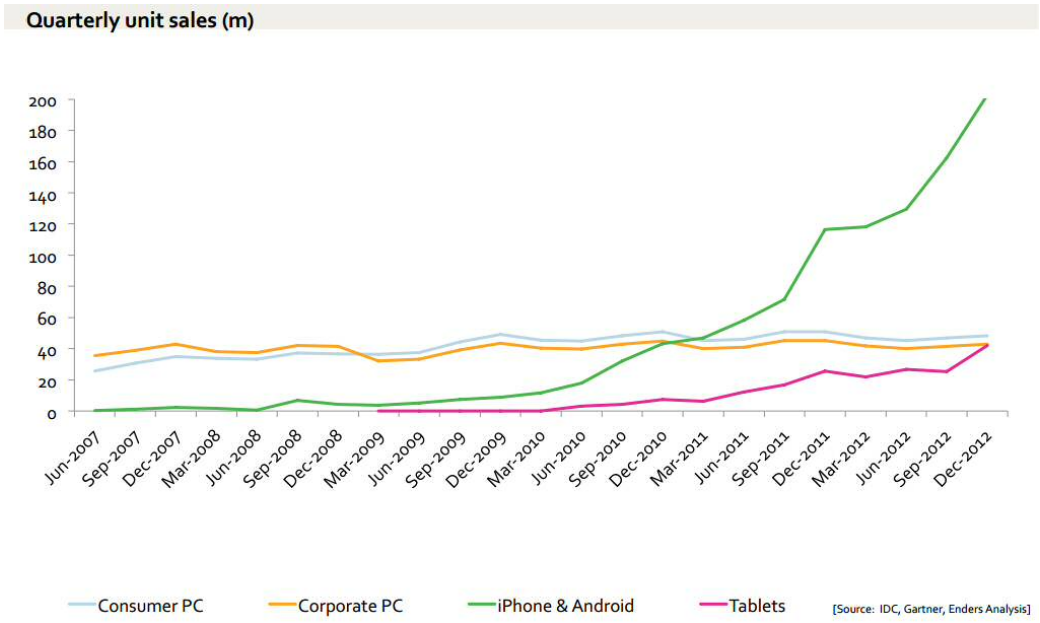

Figura B.1: Venda Unitária Mensal (em Milhões) por Produto nos Últimos Anos

Desta maneira, nota-se que enquanto os computadores, sejam eles de uso pessoal ou corporativo, não apresentam crescimento no número de vendas ao longo dos anos, os dispositivos móveis crescem aproximadamente de maneira exponencial.

Além disso, somente o número de unidades de *smartphones* está quase superando o número de computadores pessoais e corporativos (em conjunto), como pode ser verificado na figura B.2 [27].

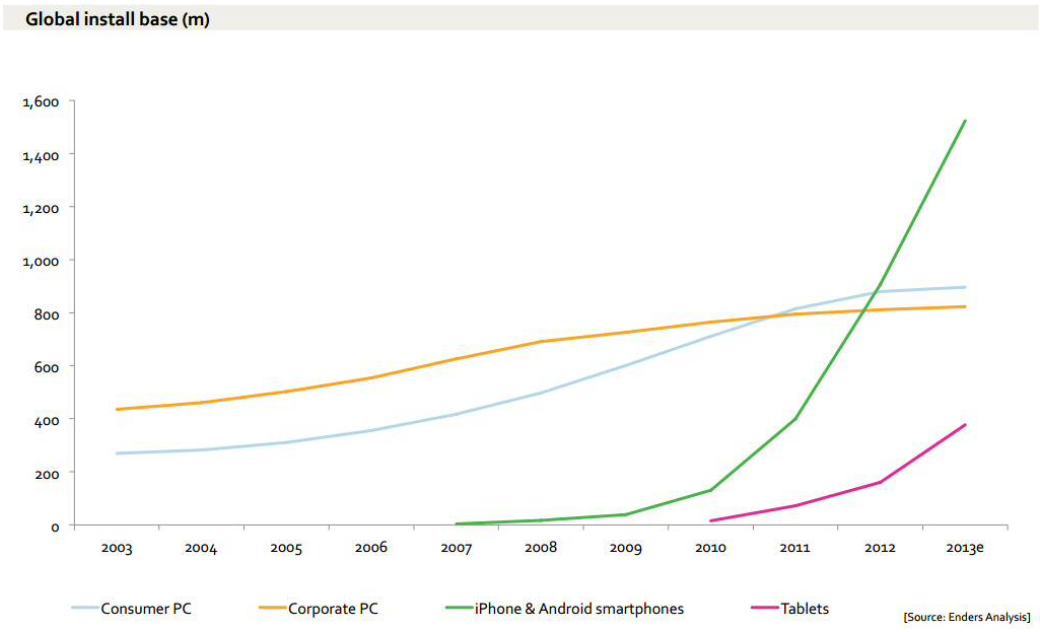

Figura B.2: Número de Unidades (em Milhões) por Produto nos Últimos Anos

Assim sendo, o número de *smartphones*, seguindo a perspectiva de crescimento obtida nos últimos anos, deve superar o de computadores ainda em 2014 [28], indicando a atual e crescente utilização destes aparelhos.

No cenário nacional Brasileiro a tendência segue a global. Em 2013 as vendas de *tablets* foi maior que a de *notebooks* e computadores de mesa [29]. Além disso, o Brasil contém o quarto maior mercado mundial de *smartphones* e está entre os 10 maiores mercados de *tablets* [30].

Estima-se que a população brasileira em 2017 será de aproximadamente 203 milhões [31] e, nesse mesmo ano, estima-se que o Brasil terá 70,5 milhões de usuários de *smartphone* [32], resultando numa concentração média de aproximadamente um *smartphone* para cada três habitantes.

Apêndice C

Programação Orientada a Objetos

Este apêndice conterá, uma ideia geral de programação Orientada a Objetos. Como a aplicação deste conhecimento varia entre as linguagens de programação, nenhuma estrutura de códigos ou comandos será introduzida, assim sendo, não será focada em um tipo de linguagem. O intuito é saber a teoria por trás deste paradigma para ser capaz de aplicá-la em qualquer linguagem. Os estudos aqui demonstrados foram baseados nas referencias [33] [34].

C.1 Objetos

A base fundamental de Orientação a Objetos são os objetos, uma vez que esta entidade contém operações para manipular suas informações.

Em linguagens de programação estruturada, variáveis são instâncias (exemplares) de uma estrutura. Já em POO, objetos são instâncias de classes.

Nos objetos são definidos quais informações (atributos) ele conterá, bem como como tais atributos poderão ser manipulados (métodos).

C.2 Atributos

Os atributos (ou variáveis membro) são os campos que armazenam as informações referentes a determinados objetos. O estado de um objeto é o conjunto de valores que seus atributos possuem em um determinado instante.

Tomando-se um automóvel como sendo o objeto, como já explicado anteriormente, esse automóvel deverá ter seus atributos dados da seguinte maneira:

Classe: Automóvel

Atributos:

Número de rodas

Tipo de Combustível

Velocidade máxima

Cor

C.3 Métodos

Os métodos são os procedimentos que permitem ao objeto realizar suas ações. Tudo que se espera que um objetos deva fazer precisa ser especificado por um método.

Continuando com o exemplo anterior:

Classe: Automóvel

Atributos:

Número de rodas

Tipo de Combustível

Velocidade máxima

Cor

Métodos:

Acelerar

Frear

Virar para esquerda

Virar para direita

C.4 Classes

Uma classe é um conjunto de objetos com características e comportamentos comuns. Seguindo o exemplo:

Classe: Automóvel

Objeto: Caminhonete

Atributos:

Número de rodas: 4

Tipo de Combustível: Diesel

Velocidade máxima: 180 km/h

Cor: Verde

Métodos:

Acelerar

Frear

Virar para esquerda

Virar para direita

Objeto: Motocicleta

Atributos:

Número de rodas: 2

Tipo de Combustível: Gasolina

Velocidade máxima: 230 km/h

Cor: Azul

Métodos:

Acelerar

Frear

Virar para esquerda

Virar para direita

Note que tanto o objeto Caminhonete, quanto o Motocicleta possuem os mesmos métodos e os mesmos atributos, contudo os valores de seus atributos são diferentes.

C.5 Abstração

A abstração, a "grosso modo", significa que só deve ser representado o que realmente for ser utilizado, deixando de representar métodos ou variáveis membro desnecessárias. Pelo exemplo da classe Automóvel, se o projeto consiste num sistema para descrever suas características básicas, não é interessante conter atributos para "Número de tomadas de 12V"ou "Numero de antenas".

C.6 Encapsulamento

O encapsulamento consiste no princípio de projeto de que cada componente de um programa deve conter todas as informações necessárias para seu completo funcionamento. Na OO, os dados (variáveis membro), e os processos (métodos) estão encapsulados numa mesma

identidade, o objeto.

O encapsulamento é também o princípio pelo qual determinado componente, seja ele um método ou uma variável membro esteja pública ou privada. Se pública, determinado componente poderá ser acessado por qualquer outra classe, se privado, esse componente apenas poderá ser acessado dentro da própria classe.

Geralmente as variáveis membro são privadas, e implementam-se métodos para modificá-las. Desta maneira, sempre que necessário alterar seu valor, isso deverá ser feito através de algum método, impedindo que, acidentalmente, alguém acesse-a diretamente, corrompendo sua informação. Desta maneira, quase a totalidade dos programas numa linguagem OO (orientada a objetos) consiste na comunicação de objetos por meio de chamadas aos seus métodos, inclusive visando a modificação de suas variáveis membro.

A ideia básica do encapsulamento é a de disponibilizar um objeto com todas as suas funcionalidades sem necessariamente saber como ele funciona, ou seja, ser capaz de reutilizar classes sem necessariamente saber seu código, o importante é conhecer suas operações disponíveis e proteger suas características por meio de acessos públicos ou privados.

C.7 Ligações e Associações

As ligações e associações são os mecanismos pelos quais é possível estabelecer interações entre objetos e classes.

Uma associação descreve um grupo de ligações com estrutura e semântica comuns. A ligação é uma conexão entre duas instâncias de objetos, ou seja, a ligação é uma instância da associação.

C.8 Generalização e Herança

Generalização e herança são abstrações de compartilhamento de similaridades entre classes, preservando suas diferenças.

A generalização consiste no relacionamento entre uma classe e uma ou mais versões refinadas (especializações) da mesma. Já a herança representa a relação entre as classes por meio de hierarquias.

A classe sendo refinada é chamada de superclasse (ou classe base), já sua versão refinada é chamada de subclasse (ou classe derivada).

Dividem-se classes em subclasses, mantendo o princípio de que cada subclasse herda as características da classe da qual foi derivada e ainda cria algumas características particulares. Novamente, exemplificando com nossa classe Automóvel:

Superclasse: Automóvel

Atributos:

Número de rodas

Tipo de Combustível

Velocidade máxima

Cor

Métodos:

Acelerar

Frear

Virar para esquerda

Virar para direita

Subclasse: Caminhão

Atributos:

Número de carretas

Métodos:

Desacoplar carreta

A subclasse Caminhão possuirá os mesmos métodos e atributos que sua superclasse Automóvel, porém ela é mais especializada, uma vez que contém atributos e métodos próprios.

Ressalta-se ainda que generalização e herança podem ser recursivamente aplicadas a um número arbitrário de níveis, e ainda, um dos principais usos de classe derivadas é aumentar a eficiência da programação pela não necessidade da criação de códigos repetitivos, ou seja, ao tornar Caminhão subclasse de Automóvel, não há necessidade de fazer novamente os métodos e atribuições já existentes na superclasse.

C.9 Polimorfismo e Sobrecarga

O polimorfismo consiste no uso de um único nome para definir várias formas distintas, em outras palavras, refere-se a criação de uma família de funções que compartilham do mesmo nome, mas cada uma tem um código independente. A sobrecarga é um tipo particular de

polimorfismo. Por exemplo, seja o operador aritmético "+"(simbolo de soma). O computador executa operações completamente diferentes para somar números inteiros e números reais, porém a soma desses dois tipos de dados é representado por um único símbolo.

C.10 Construtores e Destrutores

Quando um objeto é criado, automaticamente é feita sua inicialização através de um método chamado Construtor, tal método possui o mesmo nome da classe e é executado automaticamente toda vez que um objeto for criado. Quando um objeto é destruído, há outro método especial que é chamado automaticamente. Esta função é chamada destrutor. Ressalta-se ainda que o destrutor não pode receber argumentos nem ser chamado explicitamente pelo programador.

Apêndice D

Android

Nesta seção serão apresentadas algumas informações importantes sobre a programação para o Sistema Operacional Android. As referências [9], [35] [36] e [37] serviram como base para os estudos desenvolvidos nesta fase.

D.1 Histórico

Em agosto de 2003 foi fundada a Android Inc., uma empresa que tinha por objetivo desenvolver um pequeno sistema para celulares. Quase dois anos mais tarde (agosto de 2005), a Google anuncia a compra desta [38], dando continuidade ao desenvolvimento do produto (até então, apenas para telefones móveis), tendo como principal objetivo o de ser uma plataforma flexível, aberta e de fácil migração para os fabricantes.

Em novembro de 2007, a Google divulga a formação da OHA, *Open Handset Alliance*, que consiste na união de diversas grandes empresas (na época 34 e hoje 87) do ramo tecnológico com o propósito de padronizar um sistema único de código aberto para dispositivos móveis [39]. No mesmo dia, a OHA anunciava o Android como um novo sistema operacional para plataformas móveis.

De acordo com a OHA:

> *Hoje, existem 1,5 bilhão de aparelhos televisores em uso em todo o mundo e 1 bilhão de pessoas possuem acesso à internet. Entretanto, quase 3 bilhões de pessoas têm um telefone celular, tornando o aparelho um dos produtos de bens de consumo mais bem sucedidos do mundo. Dessa forma, construir um aparelho de celular superior melhoraria a vida de inúmeras pessoas em todo o mundo. A Open Handset Alliance é um grupo formado por empresas líderes em tecnologia*

> *móvel que compartilham essa visão para mudar a experiência móvel de todos os consumidores [40].*

Assim sendo, em setembro de 2008 surgiu a primeira versão comercial do Android, o Android 1.0. De lá para cá o sistema operacional sofreu diversas atualizações e melhorias, contando hoje com versões que atendem tanto *smartphones* quanto *tablets* [1].

Além das melhorias do sistema operacional Android e dos avanços tecnológicos de *hardware* presentes nos *smartphones* e *tablets* (mostrando-se cada vez mais portáteis e robustos) tais dispositivos estão com acelerado crescimento de vendas, chegando a ter ultrapassado as de microcomputadores em 2011 [41]. As vendas de *smartphones*, por exemplo, cresceram em média 73% de 2010 para 2011 [42].

Alavancado com esse número de vendas, o Android conseguiu uma enorme ascensão entre o mercado de sistemas operacionais de *smartphones*, tendo alcançado um crescimento de 379% em agosto de 2011 (quando comparado ao mesmo período em 2010), ocupando, atualmente, quase 80% do mercado mundial de dispositivos móveis, estando presente em 781 milhões de dispositivos [43].

Além das estatísticas comerciais, as características técnicas desta plataforma a tornam especialmente vantajosa. Dentre elas, pode-se citar sua arquitetura flexível (que permite a integração entre programas nativos e novos programas desenvolvidos por terceiros), sua utilização de kernel Linux (que permite a multitarefa, que é a capacidade de executar várias tarefas simultaneamente, dividindo o uso da memória), permissão de execução de aplicativos em segundo plano e suporte a gráficos 3D (faz uso de OpenGL, biblioteca gráfica multiplataforma) [9].

D.2 Ferramentas para Ambiente de Programação

As aplicações para o sistema operacional Android serão escritas em Java, que é uma linguagem de código aberto e orientada a objetos. O código fonte resultante é então compilado em bytecodes, gerando arquivos com extensão *.class que, em seguida, serão convertidos para o formato *.dex, que será interpretada pela Virtual Machine Dalvik (Máquina Virtual própria para executar aplicativos no Android). Os arquivos *.dex e outros recursos (como imagens e sons) serão empacotados em um único arquivo com extensão *.apk (*Android Package File*) que representa a aplicação final a ser distribuída e, consequentemente, instalada no *smartphone* ou *tablet* [9] [36].

Para realizar esse processo, é necessário preparar o ambiente de programação do Android, utilizando as seguintes ferramentas:

- **Java *Development Kit* (JDK):** O JDK pode ser encontrado no site da Oracle [44]. Como o Andoid é baseado em linguagem de programação Java, necessita-se de seu compilador.
- **Eclipse IDE:** O Eclipse IDE é o *software* responsável por executar o *plugin* ADT e proporcionar uma interface gráfica amigável para o desenvolvimento.
- **Android SDK:** O Android SDK é o kit de desenvolvimento para o Android. Ele permite implementar e trabalhar as funções existentes para o dispositivo.
- ***Plugin* ADT:** Ferramenta que faz uma espécie de link entre o Eclipse e o Android SDK, fornece o ferramental específico para a programação. Pode ser encontrado em [45].

Ressalta-se que, para desenvolver este trabalho, foi utilizado o sistema operacional Windows 7 de 64 bits, desta maneira, as especificações de instalação e configuração do ambiente de programação serão baseadas nesta plataforma.

D.3 Configuração do Ambiente de Programação

Existem duas maneiras de configurar o ambiente de programação, sendo uma delas utilizando o "ADT *Bundle*", que consiste na utilização de um pacote contendo o Eclipse IDE, o Android SDK e o *plugin* ADT, sendo uma opção rápida e prática, porém não personalizável. A outra opção é a "*Standalone*"e consiste em instalar os programas em separado, sendo assim mais demorada porém com maior personalização. Independente da maneira escolhida, a diferença durante o desenvolvimento é mínima.

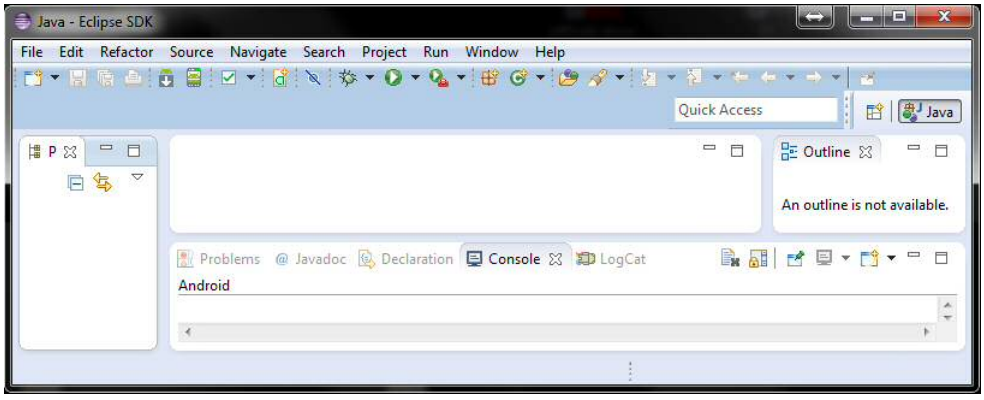

Figura D.1: Janela do Ambiente de Programação

D.3.1 ADT *Bundle*

Para a instalação utilizando o ADT Bundle, deve-se seguir os seguintes passos:

1. Realizar o download do executável do JDK, na versão que desejar [44];
2. Instalar o JDK;
3. Realizar o download do ADT Bundle, que pode ser encontrado em [46];
4. Descompactar o ADT Bundle;
5. Na pasta descompactada, basta abrir o diretório "Eclipse"e executar o mesmo;
6. O *software* que abrirá corresponde ao ambiente pronto para uso.

Apesar desta maneira ser a mais fácil, não foi a adotada no desenvolvimento deste trabalho.

D.3.2 *Standalone*

Para a instalação *Standalone*, deve-se seguir os seguintes passos:

1. Realizar o download do executável do JDK, na versão que desejar [44];
2. Instalar o JDK;
3. Realizar o download do Eclipse IDE, que pode ser encontrado em [47].
4. Instalar o Eclipse IDE;
5. Realizar o download do Android SDK, que pode ser encontrado em [46] na seção *Use an existing IDE*;
6. Instalar o Android SDK. Ao final da Instalação será oferecido para executar o SDK Manager, aqui é importante que não se execute o mesmo;
7. Instalado o Andoid SDK, abra o Eclipse IDE e selecione *Help* e em seguida, *Install New Software*;
8. Na janela que se abrir, clique em *Add*, no canto superior direito;

9. Na caixa de diálogo *Add Repository* que abriu, preencha o campo "Name"com "ADT Plugin"e o campo "URL"com "`https://dl-ssl.google.com/android/eclipse/`"e clique em "OK";
10. Na caixa de diálogo *Available Software*, selecione o checkbox de "Developer Tools"e clique em "Next";
11. Na janela seguinte será exibida uma lista de ferramentas que serão baixadas, apenas clique em "Next";
12. Leia e Aceite os termos da lincença e clique em "Finish";
13. Caso apareça um aviso de segurança dizendo que a autenticidade ou validade do *software* não pode ser estabelecida, clique em "OK";
14. Quando a intalação finalizar, reinicie o Eclipse.

Com tudo instalado, basta apenas configurar a localização do diretório onde foi instalado o SDK, de acordo com os seguintes passos:

1. Após o Eclipse ter sido reiniciado, caso apareça a janela "Welcome to Android Development", clique em "Use existing SDKs";
2. Selecione o diretório onde foi instalado o Android SDK e clique em "Next";
3. Feito isso o ambiente estará pronto para uso.

A opção *Standalone* foi escolhida para realização deste trabalho principalmente pelo fato dela permitir que se escolha qual versão do Eclipse IDE deseja-se usar, uma vez que existem dezenas de versões disponíveis, cada qual com suas peculiaridades.

D.4 Adicionando Plataformas e Pacotes

O Android SDK separa as ferramentas, plataformas e outros componentes em pacotes que podem ser baixados utilizando o Android SDK Manager. O Android SDK original que foi instalado inclui apenas algumas ferramentas de desenvolvimento. Para desenvolver um aplicativo, deve-se realizar o download de pelo menos, uma versão da plataforma Android e a última versão do SDK *Platform-tools*. Ressalta-se que, independente de como foi feita a configuração do ambiente de programação (ADT *Bundle* ou *Standalone*) estes pacotes devem ser adicionados.

Para isto, deve-se seguir os passos:

1. Execute o SDK Manager através do arquivo "SDK Manager.exe"presente na raiz do diretório onde foi instalado o Android SDK;

2. Após aberto, o SDK Manager exibira todos os pacotes e plataformas disponíveis. Como mínima configuração deve escolher: "*Tools*", alguma versão do Android (preferencialmente as mais recentes) e "Android *Support Library*".

3. Após escolhidos os pacotes, clique em "*Install*";

4. Finalizada a instalação, tudo que é necessário para desenvolver um aplicativo estará instalado e configurado

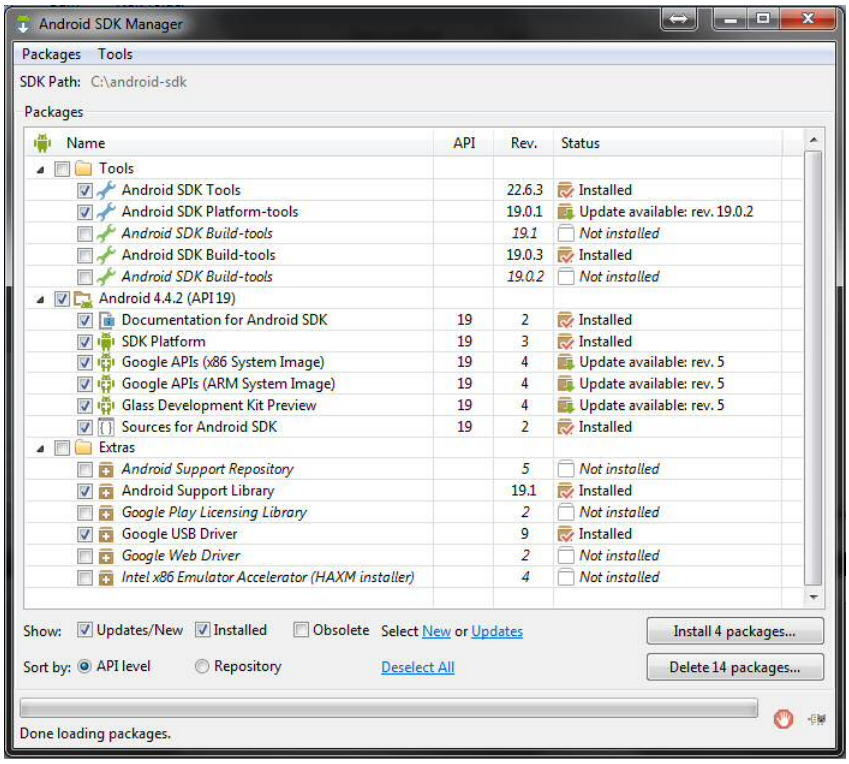

Figura D.2: Janela do SDK Manager

D.5 Versões do Android

Atualmente a última versão do Android lançada é a 4.4, intitulada de KitKat, contudo, como o intuito de um *software* é ser o mais compatível possível com os *hardwares* existentes, não seria vantajoso fazer um aplicativo para uma parcela tão pequena de *smartphones* (esta versão

foi lançada no final de Novembro e corresponde a apenas a 8,5% dos aparelhos que possuem o Andoid como SO). Dessa maneira, o mais lógico é desenvolver uma aplicação que busque executar na maioria dos dispositivos.

A versão mais compatível em execução, é a 2.2, "Froyo", entretanto ela se mostra já defasada em relação às versões mais atuais, ou seja, de nada adianta ter um produto com muita compatibilidade, porém com pouca velocidade de execução e desatualizado.

Contudo, existe algo muito interessante no desenvolvimento do Android, que é poder desenvolver para uma versão qualquer *x* e manter seu *software* compatível com qualquer versão posterior a versão *y* qualquer, ou seja, pode-se desenvolver para a versão 4.4, utilizando assim todos os recursos de segurança e funcionalidades desta versão, porém mantendo o código compatível com qualquer versão posterior a 2.2, garantindo assim sua globalidade de execução. Desta maneira o programa funcionará em praticamente 100% dos dispositivos existentes e conterá a robustez de código das versões mais modernas.

A figura D.3

Version	Codename	API	Distribution
2.2	Froyo	8	1.0%
2.3.3 - 2.3.7	Gingerbread	10	16.2%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	13.4%
4.1.x	Jelly Bean	16	33.5%
4.2.x		17	18.8%
4.3		18	8.5%
4.4	KitKat	19	8.5%

Data collected during a 7-day period ending on May 1, 2014.
Any versions with less than 0.1% distribution are not shown.

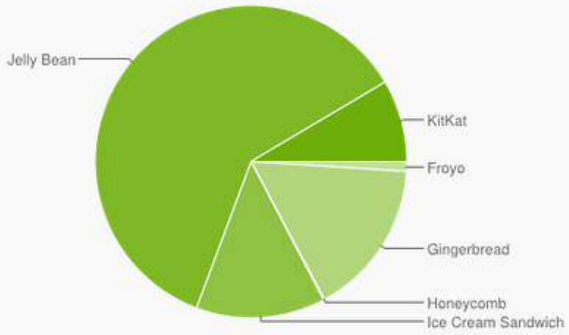

Figura D.3: Distribuição das Versões do Android (Dados Coletados em Abril e Maio de 2014) [1]

D.6 Emulador Android SDK

O Android SDK possui um emulador de um dispositivo móvel que pode rodar no computador a fim de possibilitar a prototipagem, desenvolvimento e teste, muitas vezes sem a necessidade de um dispositivo físico.

Basicamente todas as características de *hardware* e *software* de um dispositivo típico, ex-

ceto o fato de não realizar chamadas, acesso ao acelerômetro e *bluetooth*, entre outros, são emuladas. Para isto, o emulador utiliza as configurações do Android Virtual Device (AVD) para permitir ao usuário configurar até mesmo alguns aspectos de *hardware* do telefone emulado. O emulador inclui também funcionalidades extras para fazer o *debug* do código ou até mesmo simula latências e queda de sinal da operadora.

Uma vez tendo o código elaborado no Eclipse IDE, basta clicar em "*Run*", escolher o AVD criado e aguardar o carregamento do emulador e a respectiva execução do programa desenvolvido. A figura D.4 exibe a interface do emulador.

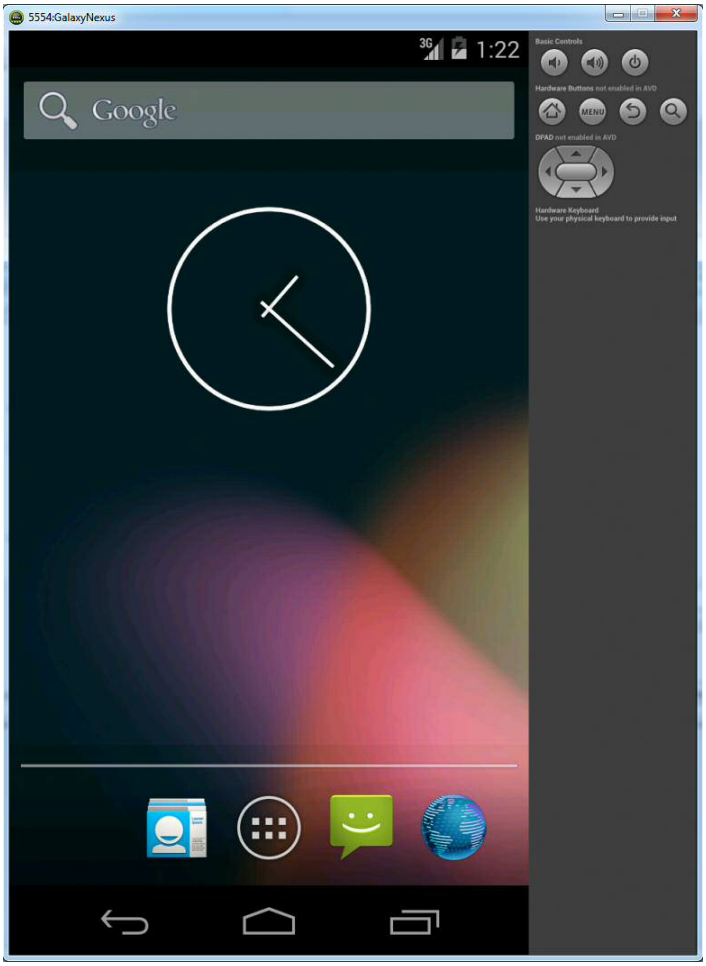

Figura D.4: Interface do Emulador Android, emulando um Smartphone modelo Nexus 4

D.7 Criando e Configurando um AVD

Para poder utilizar o emulador, deve-se antes criar um AVD. Para isto, basta seguir os seguintes passos:

1. Execute o AVD Manager através do arquivo "AVD Manager.exe"presente na raiz do

diretório onde foi instalado o Android SDK;

2. Na janela aberta, na aba "Android *Virtual Devices*", clique em "*New*";

3. Na nova janela que abriu, basta nomear o AVD que deseja criar, preencher as informações exigidas (modelo do dispositivo e versão do Android a ser executada, por exemplo) e clicar em "OK";

4. Feito isso o AVD criado será exibido na janela principal do AVD Manager e o mesmo está pronto para ser executado através do Eclipse IDE.

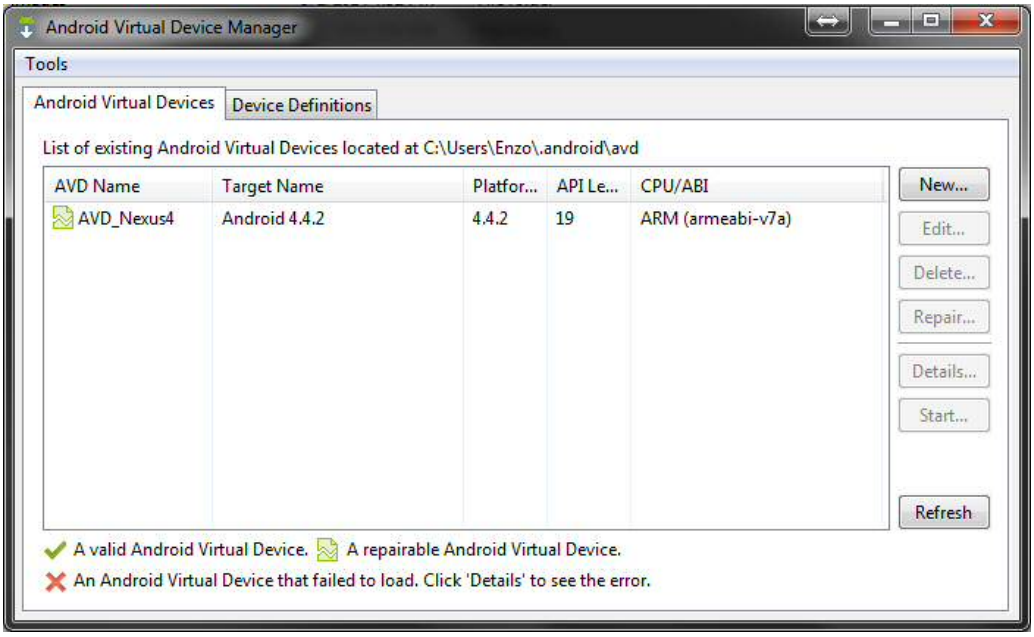

Figura D.5: Janela Principal do AVD, Contendo um AVD Criado que Emulará o *Smartphone* modelo Nexus 4

D.8 Arquitetura do Android

O SO Android pode ser designado como uma pilha de *softwares* em diferentes camadas, em que cada camada consiste num agrupamento de diversos componentes de programa. Tais camadas compõe o sistema operacional, *middleware* e aplicativos. Cada camada da arquitetura fornece diferentes serviços para a camada acima dela. A figura D.6 representa tais camadas, os detalhes das mesmas serão dados em seguida [48] [9] [49]:

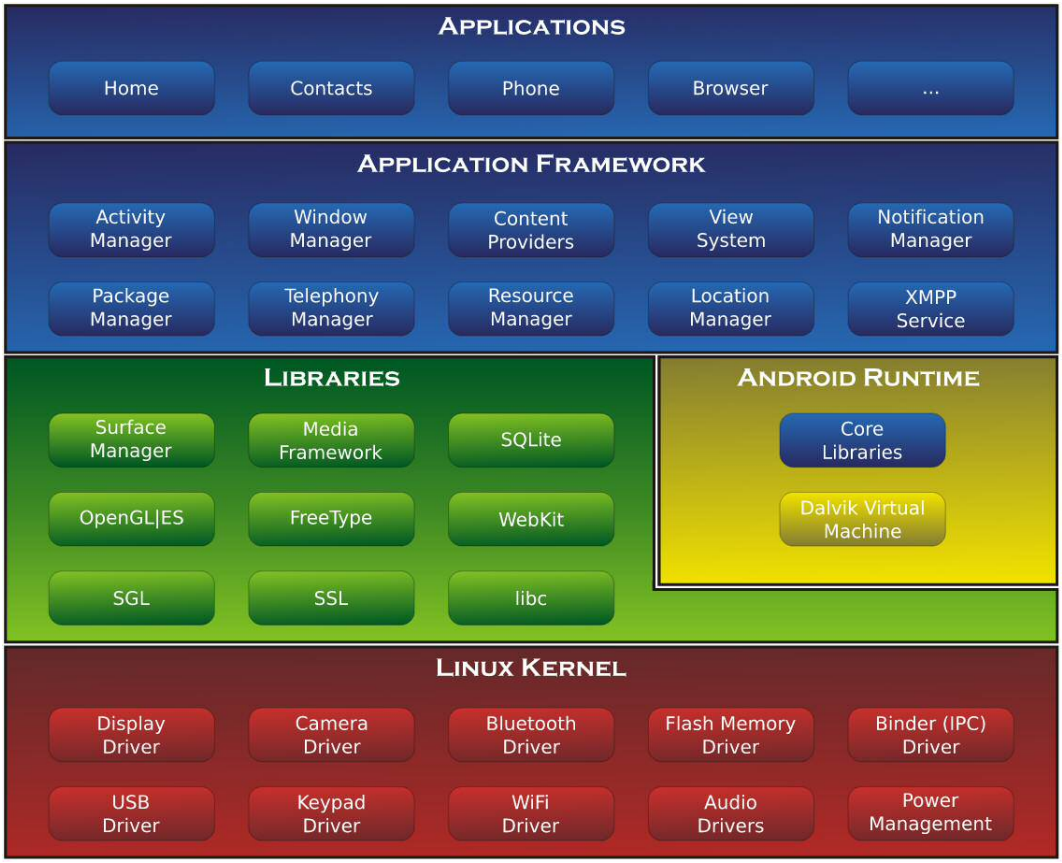

Figura D.6: Arquitetura do Sistema Operacional Android

D.8.1 Nível Zero - Linux *Kernel*

No chamado "nível zero", tem-se a base da pilha, ou seja, o *Kernel* (Linux *Kernel*). Sua primeira versão foi desenvolvida sobre a versão 2.6 do Sistema Operacional Linux. É no *Kernel* que se localiza os *softwares* de gerenciamento de memória, configurações de segurança e de rede, entre outros, atuando também como responsável pela abstração entre o *hardware* e os aplicativos (*drivers* de *hardware*).

D.8.2 Nível Um - *Libraries* e Android *RunTime*

No nível um encontram-se as camadas de bibliotecas (*Libraries*) e de tempo de execução (Android *RunTime*).

A camada de biblioteca consiste num conjunto de instruções que informam ao dispositivo como lidar com diferentes tipos de dados, incluindo um conjunto de biblioteca C e C++ usadas por diversos componentes do sistema. Essas bibliotecas são expostas para os programadores via códigos na linguagem Java, através do *framework* do sistema, compondo assim

a estrutura de aplicativo Android. Ressalta-se ainda que são essas as bibliotecas responsáveis por prover funcionalidades para manipulação de áudio, vídeo, gráficos, banco de dados, navegador, entre outros.

A camada de tempo de execução inclui um conjunto de bibliotecas do núcleo Java (*Core Libraries*), permitindo que cada processo rode sua própria instância na maquina virtual. Desta maneira, para desenvolver aplicações para o Android, é utilizada a linguagem de programação Java.

Embora o desenvolvimento de aplicativos seja realizado através da linguagem Java, as aplicações não são executadas em uma máquina virtual Java tradicional (não existe uma JVM (Java Virtual Machine) inclusa no sistema), mas sim numa máquina virtual própria, denominada Dalvik (DVM), que é otimizada especialmente para e execução de aplicativos em dispositivos moveis, executando seu próprio tipo de *bytecodes*.

Essa maquina virtual foi construída por engenheiros da Google, visando um consumo mínimo de memória e isolamento de processos. Ela se torna essencial pois permite que múltiplas instâncias executem ao mesmo tempo, de modo que se uma aplicação parar, ela não afeta quaisquer outras aplicações rodando no dispositivo, além disso, como a máquina virtual está baseada em registradores ela é desenvolvida de forma otimizada para requerer pouca memória.

D.8.3 Nível Dois - *Application Framework*

No nível dois, encontra-se a camada de *Application Framework*, ou *framework* de aplicação, que disponibiliza aos desenvolvedores as mesmas APIs utilizadas para criar as aplicações originais do sistema, funcionando como um meio de ligação com a camada de bibliotecas. Este *framework* permite que o desenvolvedor tenha o mesmo acesso ao sistema, que os aplicativos da camada de aplicativos possuem. No geral, este *framework* foi desenvolvido para abstrair a complexidade e simplificar o reuso de procedimentos.

Entre as APIs disponíveis, encontram-se, por exemplo, a *Location Manager* (usada para obter a posição geográfica do usuário via GPS), *Telephony Manager* (usada para prover informações sobre o dispositivo, como bateria e serviços de telefonia), *Window Manager* (responsável pelo gerenciamento de toda janela de uma aplicação), *Notification Manager* (permite que uma aplicação exiba notificações, controle LEDs, sons e vibração) e *Activity Manager* (responsável pelo gerenciamento de cada atividade do sistema).

No Android cada atividade é gerenciada através de uma pilha de atividades. Toda nova

atividade criada vai para o topo de pilha de atividades e se torna uma *running Activity*, o que quer dizer que a mesma será executada (nas seções seguintes serão dados maiores detalhes sobre as *Activities*).

D.8.4 Nível Três - *Applications*

Neste nível encontra-se a camada de aplicações e as funções básicas do dispositivo, sendo a camada mais alta da arquitetura do sistema. Esta é a camada de interação entre o usuário e o dispositivo móvel, e é composta pelo conjunto de aplicativos nativos do sistema como, por exemplo, os clientes de e-mail, calendário, calculadora, mapas, navegadores de internet, jogos, e outros.

D.9 Estrutura de Um Projeto

Todo novo projeto Android possui uma estrutura de pastas especial, de modo a aproveitar o uso dos *resources* (recursos) e proporcionar uma organização para o conteúdo do programa. A figura D.7 mostra a estrutura geral que todo novo projeto Android possui, seus principais componentes serão detalhados em seguida [9] [50] [51].

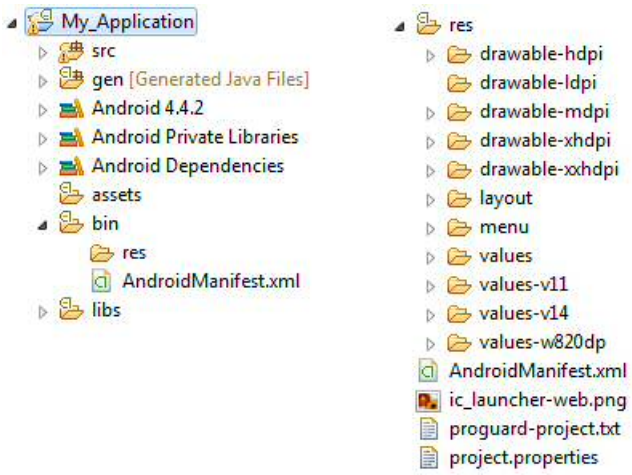

Figura D.7: Estrutura Geral de um Projeto Android

D.9.1 Diretório src

O diretório src (*Source*) da estrutura do projeto possui todas as classes escritas em Java, ou seja, todos os arquivos de código fonte (.java) desenvolvido, bem como os pacotes que os organizam.

D.9.2 Diretório gen

O diretório gen (*Generated Source*) é onde encontram-se as classes geradas automaticamente pelo SDK do Android. Neste diretório encontram-se classes que o Android usa internamente para executar o projeto.

Como exemplo pode-se citar a classe R.java (classe onde são mapeadas as constantes de acesso aos recursos do aplicativo) e a classe BuildConfig.java (classe onde são definidas algumas configurações da construção do projeto).

Ressalta-se ainda que o conteúdo deste diretório é gerado automaticamente e não deve, de forma alguma, ser alterado.

Um fato interessante é que a classe R.java contém IDs para todos os itens da pasta res, e é através desses IDs que pode-se usar todos os *resources* na nossa aplicação, em outras palavras, é a classe que faz a interface entre código fonte e *layout* da aplicação.

D.9.3 Diretório res

O diretório res (*Resources*) é o local onde encontram-se os recursos da aplicação. Dentro deste diretório, existem subdiretórios para cada tipo de recurso, como:

- res/drawable-<screen cfg>: Local onde deve-se colocar os arquivos de imagens (png, jpeg ou gif). A divisão entre hdpi (*High* DPI), mdpi (*Medium* DPI), ldpi (*Low* DPI) serve para guardar as imagens em resoluções diferentes;

- res/layout: Local onde ficam os arquivos XML que representam o *layout* das *Activities* (telas);

- res/values: Local onde ficam os arquivos XML com as mensagens do aplicativo (já com suporte a internacionalização, que é um recurso para facilitar tradução das mensagens).

A arquitetura de interface gráfica do Android é baseada no padrão de projetos MVC (*Model - View - Controller*), onde a camada *Model*, é responsável pelas regras da aplicação, a camada de *View*, é responsável pela interface gráfica e a camada de *Controller* gerencia os eventos de uma determinada tela.

A grande vantagem proporcionada por este padrão é a modularidade, em termos práticos, consegue-se alterar todo *layout* de uma determinada tela sem mexer no código que diz respeito ao comportamento da mesma. Desta maneira, a cor da tela, o tamanho e a fonte dos

botões, e tudo que diz respeito a interface gráfica é definido na camada *View*, enquanto o que irá acontecer quando os botões forem clicados será ministrado pela camada de *Controller*.

Desta maneira, os *layouts* XML assumem o papel de *View*, as *Activities* assumem o papel de *Controller*, tudo isso enquanto a classe R.java, faz a ligação entre as duas camadas, permitindo o acesso de recursos encontrados na pasta "res"nas *Activities* do projeto.

D.9.4 Arquivo AndroidManifest.xml

O AndroidManifest.xml é um arquivo XML que armazena as configurações necessárias para a execução do aplicativo para Android. É neste arquivo que estarão configurados os componentes da aplicação, o nome do pacote padrão do projeto, o nome das *Activities* usadas no aplicativo, a versão da API do Android que o aplicativo irá suportar, a versão do próprio aplicativo, bem como as permissões que o aplicativo precisará para executar no Android.

D.10 Ciclo de Vida de Uma Activity

Activity é uma classe que representa uma tela da aplicação Android, sendo responsável por controlar os eventos da tela, definir a interface gráfica do usuário, bem como tratar eventos gerados nesta tela como, por exemplo, um clique na tela, botão pressionado e escolha de item no menu.

O clico de vida de *Activities* é similar ao ciclo de vida de *Threads*. É composto por 7 métodos que indicam seu estado atual: onCreate(), onStart(), onResume(), onPause(), onStop(), onRestart() e onDestroy().

Apenas através do entendimento deste ciclo de vida pode-se desenvolver aplicações mais robustas.

A figura D.8 exibe um diagrama do Ciclo de Vida de uma *Activity*.

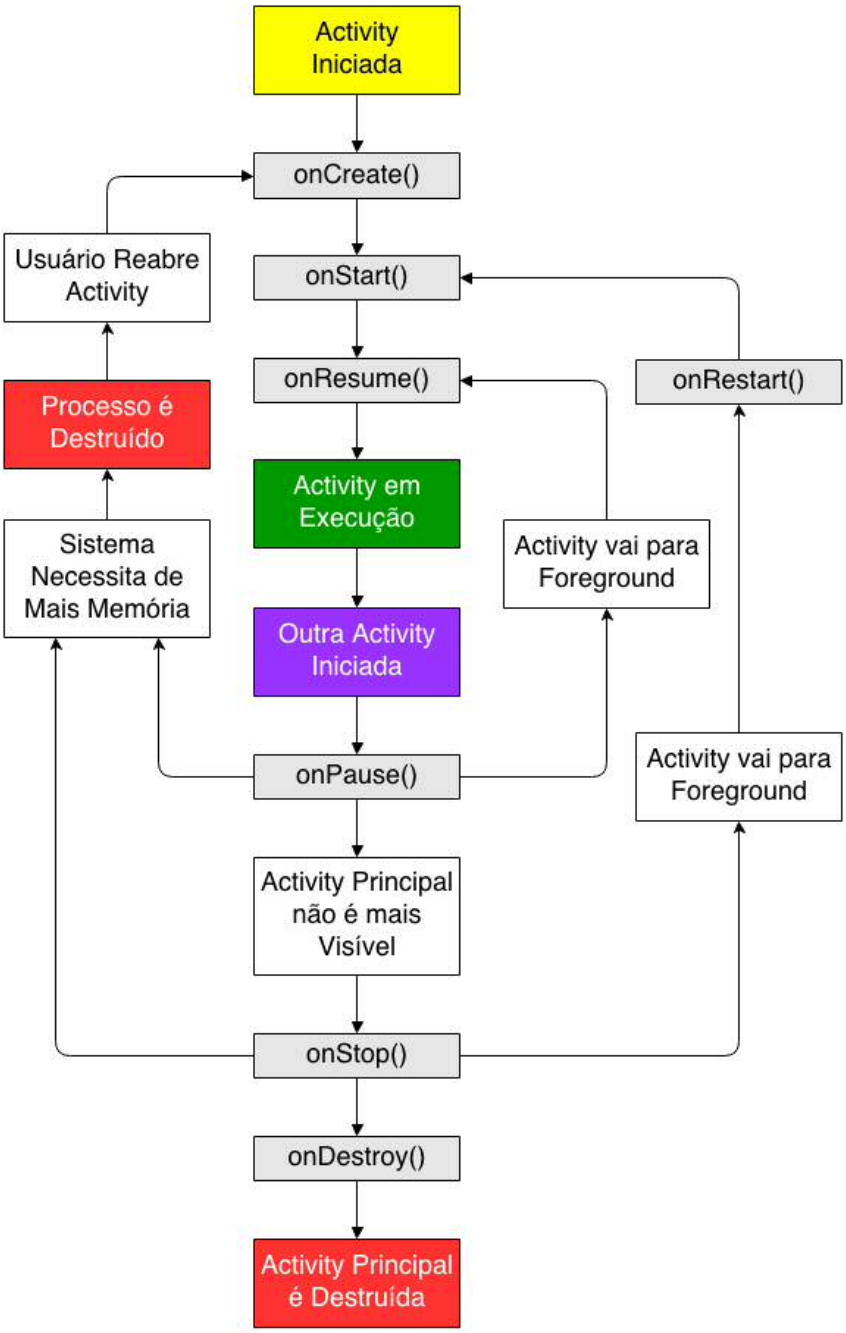

Figura D.8: Ciclo de Vida de Uma Activity

Desta maneira, os métodos significam:

- ***onCreate()***: É obrigatório e consiste na primeira função a ser executada em uma *Activity*. Geralmente é a responsável por carregar os *layouts* XML e outras operações de inicialização. É executada apenas uma vez. Após a chamada do método onCreate(), o método onStart() é chamando para iniciar o ciclo de vida da *Activity*.

- ***onStart()***: É chamada imediatamente após o método onCreate() ou onRestart() (quando

uma *Activity* que estava em *background* volta a ter foco)

- ***onResume()***: Assim como a onStart(), é chamada na inicialização da *Activity* e também quando uma *Activity* volta a ter foco. Entretanto, a onStart() só é chamada quando a *Activity* não estava mais visível e volta a ter foco, a onResume() é chamada quando a *Activity* foi completamente encoberta por outra *Activity* e volta a ter foco.

- ***onPause()***: É a primeira função a ser invocada quando a *Activity* perde o foco (isso ocorre quando uma nova *Activity* é iniciada como, por exemplo o dispositivo entrar em espera para economizar energia). Este método salva o estado da *Activity* para que possa ser recuperado no método onResume().

- ***onStop()***: Só é chamada quando a *Activity* fica completamente encoberta por outra *Activity*. Depois de parada, a *Activity* pode ser reiniciada com o método onRestart().

- ***onDestroy()***: A última função a ser executada. Depois dela, a *Activity* é considerada "morta", ou seja, não pode mais ser relançada. Se o usuário voltar a requisitar essa Activity, um novo objeto será construído. Este método pode ser chamado pelo próprio SO, para liberar recursos.

- ***onRestart()***: Chamada imediatamente antes da onStart(), quando uma *Activity* volta a ter o foco depois de estar em background.

Segundo a documentação do Android há ainda três subníveis do ciclo de vida principal:

- ***Entire Lifetime***: Ocorre apenas uma vez e define o tempo de vida completo de uma *Activity*. Acontece entre as chamadas do método onCreate() e onDestroy().

- ***Visible Lifetime***: Ocorre entre os métodos onStart() e onStop(). Durante esse período é possível visualizar a *Activity* na tela, mas ela pode não estar em primeiro plano e interagindo com o usuário. Ou seja, possa ser que outra *Activity* esteja no topo da pilha de execução.

- ***Foreground Lifetime***: Este ciclo ocorre entre os métodos onResume() e onPause() e, durante esse tempo, a *Activity* está no topo da pilha de execução e interagindo com o usuário.

D.11 Dinâmica de um Aplicativo

D.11.1 Ciclo de vida dos processos em Android

O Android tenta manter o processo relativo a uma aplicação pelo tempo necessário, mas eventualmente precisa remover processos antigos para conseguir alocar memória para novos processos que estão chegando. Para determinar quais processos permanecerão e quais serão mortos, o sistema determina uma hierarquia de importância para cada processo.

Essa hierarquia é baseada nas componentes que estão rodando no processo e o estado de cada componente. Processos de mais baixa importância são eliminados primeiro, depois o mesmo ocorre com aqueles com as seguintes mais baixas notas e assim por diante, enquanto for necessário liberar mais recursos do sistema [52].

D.11.2 *Threads* em Android

Quando uma aplicação é lançada, o sistema cria uma *thread* para execução da aplicação: *main thread*. Essa *thread* é muito importante, pois tem a responsabilidade de despachar eventos registrados pelos diferentes componentes da interface gráfica.

Contudo, quando a aplicação lida com processamento intensivo em resposta à interação com o usuário, esse modelo de *thread* única pode resultar em baixa performance se não for implementado da maneira correta. Ou seja, como tudo está ocorrendo na *thread* principal, lidar com operações de longa duração faz com que nenhum outro evento possa ser despachado. Pela perspectiva do usuário, a aplicação parece travar. Pior, se a *thread* principal ficar bloqueada por mais de 5 segundos, o usuário verá o infame diálogo "aplicativo não está respondendo". O usuário pode então decidir parar de usar a aplicação e desinstalá-la de seu dispositivo. Nesta situação, é necessário o uso de *threads* trabalhadoras [52].

D.11.3 *Threads* Trabalhadoras

Diante do desafio de trabalhar com o modelo de *thread* única descrito acima, é vital para a manutenção do tempo de resposta de sua aplicação que a *thread* principal não seja bloqueada. Se houver operações não instantâneas a serem processadas, tenha certeza que você não lidará com elas na *thread* principal. Novas *threads* paralelas devem lidar com esse tipo de trabalho.

D.11.4 Eventos e Programação Orientada a Eventos

Em geral, eventos são manifestações ou ações decorridas interação do usuário com a aplicação. Por exemplo, o click em um botão é um evento. O programa automaticamente reage à esse evento executando alguma ação, por exemplo, mudando a cor da tela. Todos os programas com interface gráfica e a maioria dos programas que implementam troca de dados através da rede ou sensores de dados (internet, *bluetooth*, GPS, entre outros) são programas orientados a eventos.

Programas orientados a eventos precisam de basicamente 2 componentes [53]:

- *Event queue*: é uma fila em que são registrados os eventos que foram iniciados pela interação entre a aplicação e o usuário;

- *Event loop*: é aqui que o programa reage aos eventos iniciados. O "*event loop*"é um *loop* infinito que espera a chegada de algum evento, checa o tipo de evento, e reage sobre o evento de um tipo específico.

Em Android, existe basicamente uma única *thread* que é responsável pela execução do *event loop* e das respostas aos eventos, ela é chamada *Main Thread* ou UI *Thread*. Porém, como visto anteriormente, o tratamento dos eventos não pode fazer com que a resposta ao usuário seja lenta. Assim, os eventos devem ser lidos rapidamente da *Event Queue*, e a UI *thread* só pode passar um tempo bem curto executando as respostas aos eventos gerados.

A solução para este problema está na criação e utilização de *threads* concorrentes ou *background threads*, cuja execução é feita totalmente em paralelo à execução da UI *thread* ou *thread* principal. Assim, essas *threads* concorrentes são criadas especialmente para lidar com operações que consomem mais tempo de processamento. Após ter criado a *thread* concorrente, a UI *thread* volta para a execução do *event loop*[53].

Apêndice E

UML - Unified Modeling Language

A descrição, exibição e os conceitos básicos, sintaxes e significados de alguns diagramas de UML serão detalhados neste apêndice. Os estudos aqui demostrados foram baseado nas referências [34] [54].

E.1 Diagramas de Classes

Um diagrama de classe mostra uma estrutura estática das classes que constituem o sistema, bem como o relacionamento entre diversas classes.

E.1.1 Representação de uma Classe

Veja a figura E.1 que representa a classe Automóvel (vide Apêndice A para maiores detalhes sobre esta classe usada para exemplos).

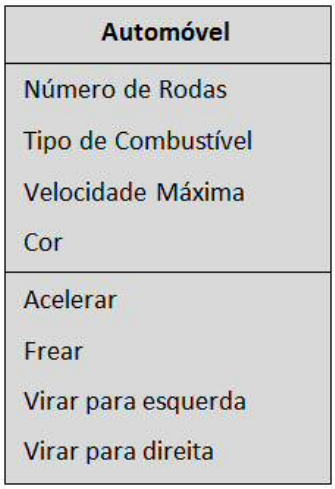

Figura E.1: Representação da classe Automóvel

A figura E.1 é composta por 3 retângulos. O primeiro deles contém o nome da classe e deve estar em negrito. Já o retângulo intermediário contém os atributos da classe. No último retângulo, deve haver os métodos que podem ser executados pela classe.

E.1.2 Associação Simples

A Associação Simples mostra o relacionamento ou dependência de uma classe com a outra. O símbolo que a representa pode ser verificado na figura E.2.

Figura E.2: Representação da Associação Simples

E.1.3 Generalização e Herança

A Herança ou Generalização indica que o comportamento da classe ou característica da classe muda. O símbolo para a Herança pode ser vista na figura E.3. Um exemplo de sua utilização se mostra na figura E.4, utilizando o exemplo da superclasse Automóvel e subclasse Caminhão, presentes no Apêndice A.

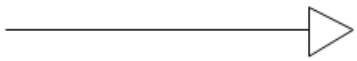

Figura E.3: Representação da Herança

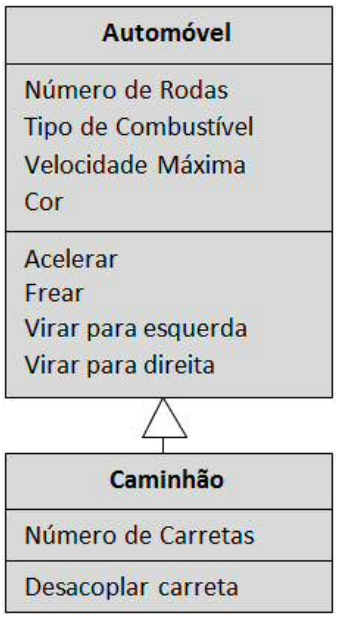

Figura E.4: Exemplo de Herança entre a superclasse Automóvel e a subclasse Caminhão

E.1.4 Agregação

Pode-se dizer que a agregação é uma associação em que um objeto é parte de outro, de tal forma que a parte pode existir sem o todo. Isso quer dizer que é uma relação do tipo "todo/parte"ou "possui um". É uma forma especializada de associação na qual um todo é relacionado com suas partes. Na figura E.5 pode-se verificar seu símbolo.

Figura E.5: Representação da Agregação

E.1.5 Composição

Tipo de associação de agregação, mas a diferença entre elas é que a composição faz parte do todo, e ainda, depende do todo. Em outras palavras, os objetos são inseparáveis, quando um objeto Pai (pertencente a uma Superclasse) é destruído o mesmo ocorre com o objeto Filho (pertencente a subclasse). Na figura E.6 pode-se verificar seu símbolo.

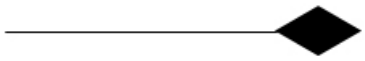

Figura E.6: Representação da Composição

E.2 Diagrama de Caso de Uso

Primeiramente, sabe-se que um Ator pode ser uma pessoa, um sistema, ou uma entidade externa. É ele quem realiza uma atividade e sempre atua sobre um caso de uso.

Um caso de uso é a descrição de uma funcionalidade (um uso específico) do sistema. Tal descrição dos usos é normalmente feita através de uma planilha contendo textos explicativos.

Em suma, os diagramas de caso de uso mostram um certo número de atores externos e suas conexões com os casos de uso que o sistema provê. Já a funcionalidade e o fluxo dos mesmos são apresentados através de diagramas de atividades

E.3 Diagrama de Atividade

Um diagrama de atividade mostra um fluxo sequencial de ações em um único processo, sendo que as ações são as unidades básicas de uma atividade. Também é verificado como uma

atividade depende da outra.

Esse diagrama pode especificar mensagens e objetos sendo enviados ou recebidos como parte das ações que estão sendo executadas. Condições e decisões, como também execução de ações paralelas, podem ser vistas neste diagrama.

E.4 Diagrama de Sequência

Consiste em um diagrama que tem o objetivo de representar como as mensagens entre os objetos são trocadas no decorrer do tempo para a realização de uma operação, ou seja, mostra a evolução de uma dada situação em determinado momento do *software*.

Apêndice F

Computação Na Nuvem

O conceito de computação na nuvem (em inglês, *cloud computing*) refere-se à utilização das mais variadas aplicações através de serviços que poderão ser acessados de qualquer lugar do mundo, a qualquer hora, e independente da plataforma, sem a necessidade de instalação de programas ou de armazenamento de dados localmente, porém com a mesma facilidade.

O acesso a tais programas, serviços e arquivos é realizado de maneira remota, utilizando da memória e da capacidade de armazenamento e cálculo de computadores e servidores compartilhados e interligados por meio da Internet. A alusão deste conceito ao termo "nuvem"se dá pelo fato de que, em fluxogramas e diagramas, a internet é representada pelo símbolo de uma nuvem [55] [56].

Apesar do conceito de nuvem de dados ser recente, esta tecnologia vem sendo usada a um bom tempo, como por exemplo, através das contas de e-mails. Os usuários que possuam um e-mail podem acessa-lo em qualquer lugar, não necessitando de um programa para isto, ou seja, basta logar na conta através do navegador de qualquer computador conectado a internet.

F.1 Tipologia

Atualmente, pode-se dividir a computação na nuvem por tipos de serviço [57] [58] [59], dentre eles, os mais comuns são:

- **SaaS - *Software* como Serviço (*Software as a Service*)**: Consiste no uso de um *software* no qual o mesmo é oferecido como um serviço. Geralmente paga-se um valor periódico referente aos recursos utilizados e/ou pelo tempo de uso. Como exemplo, pode-se citar o Google Docs [60] e o Microsoft SharePoint Online [61].

- **PaaS - Plataforma como Serviço (*Plataform as a Service*)**: Trata-se de uma solução mais ampla para determinadas aplicações, que reúne diversos recursos necessários à operação, como armazenamento, banco de dados, escalabilidade (aumento automático da capacidade de armazenamento ou processamento), suporte a linguagens de programação, segurança, entre outros. Como exemplo, cita-se o Microsoft Azure [62].

- **IaaS - Infra-estrutura como Serviço (*Infrastructure as a Service*)**: Similar ao PaaS, contudo focando em estrutura de *hardware* ou máquinas virtuais, através da utilização de uma porcentagem de um servidor, permitindo inclusive acesso ao sistema operacional, geralmente com configuração que se adequam à necessidade da aplicação. Um bom exemplo é o Amazon EC2 [63]

- **DevaaS - Desenvolvimento como Serviço (*Development as a Service*)**: Resume-se em ferramentas de desenvolvimento, muitas vezes compartilhadas. Neste segmento, têm-se como exemplo o Koding [64].

- **CaaS - Comunicação como Serviço (*Communication as a Service*)**: Consiste no uso de uma solução de comunicação unificada, que pode ser tanto hospedada no *data center* do provedor quanto do fabricante. Um bom exemplo é o Microsoft Lync [65].

- **DBaaS - Banco de dados como Serviço (*Database as a Service*)**: Esta modalidade baseia-se no fornecimento de serviços para armazenamento e acesso de volumes de dados. Um ótimo exemplo é o Parse [66], inclusive utilizado no desenvolvimento deste trabalho de conclusão de curso.

F.2 Modelos de Implantação

A Computação na Nuvem ainda pode ser classificada por seu modelo de implantação, dependendo da sua restrição ou abertura de acesso [67] [57].

- **Nuvens Privadas**: São aquelas construídas exclusivamente para um único usuário (uma empresa, por exemplo). A infraestrutura utilizada pertence ao usuário, possuindo total controle sobre como as aplicações são implementadas na nuvem. Uma nuvem privada é, em geral, construída sobre um data center privado.

- **Nuvens Públicas**: São aquelas executadas por terceiros. As aplicações de diversos usuários ficam misturadas nos sistemas de armazenamento. A implementação de uma

nuvem pública considera questões fundamentais, como desempenho e segurança, de modo que a existência de outras aplicações sendo executadas na mesma nuvem permanece transparente tanto para os prestadores de serviços como para os usuários.

- **Nuvens Comunitárias**: A infraestrutura de nuvem é compartilhada por diversas organizações e suporta uma comunidade específica que partilha as preocupações (por exemplo, a missão, os requisitos de segurança, política e considerações sobre o cumprimento). Pode ser administrado por organizações ou por um terceiro e pode existir localmente ou remotamente.

- **Nuvens Híbridas**: Consiste na composição dos modelos de nuvens públicas e privadas. Elas permitem que uma nuvem privada possa ter seus recursos ampliados a partir de uma reserva de recursos em uma nuvem pública. Essa característica possui a vantagem de manter os níveis de serviço mesmo que haja flutuações rápidas na necessidade dos recursos.

F.3 Vantagens e Desvantagens

Dentre as vantagens da computação na nuvem [68] [67] [57], pode-se citar como as mais relevantes:

- Na maioria dos casos, o acesso a determinadas aplicações é independente do sistema operacional ou *hardware* do usuário;

- Despreocupação com a estrutura, uma vez que o *hardware*, rotinas de *backup*, controle de segurança, manutenção, entre outros, são responsabilidades do fornecedor do serviço;

- Maior facilidade no compartilhamento de dados e em trabalho colaborativo. Todos os usuários acessam as aplicações e os dados da nuvem, independente de sua localização física;

- Grande disponibilidade: geralmente se um servidor parar de funcionar, por exemplo, os demais que fazem parte da estrutura continuam a oferecer o serviço, existindo assim grande redundância;

- Controle de gastos eficientes: diversas aplicações em *cloud computing* são gratuitas e, quando pagas, o usuário só o fará em relação aos recursos que usar ou ao tempo que utilizar;

- Escalabilidade: O provisionamento dinâmico de recursos sob demanda, com mínimo de esforço por parte do usuário;

- Transparência: O usuário não necessita conhecer toda a estrutura que há por trás de sua nuvem, ou seja, não precisa saber quantos servidores executam determinada ferramenta, quais são as configurações de *hardware* utilizadas, como o escalonamento é feito, ou onde está a localização física do data center.

Apesar das grandes vantagens, assim como todo sistema, a computação na nuvem também possui suas desvantagens, sendo que a maior delas vem fora do propósito da mesma, ou seja, o acesso a internet. Caso o acesso seja perdido, basicamente todos os sistemas embarcados que a utilizam serão impactados. Além disso, pode-se ainda citar [68] [67] [57]:

- Menos proteção à privacidade sob os olhos da lei, uma vez que a obtenção de informações na nuvem pode ser feita, em alguns países, inclusive sem o conhecimento e consentimento do usuário, o que não ocorre em sistemas *off-line*.

- Frágeis sistemas de segurança são fáceis de invadir, ou seja, a robustez em segurança é um fator importante e que pode ser comprometido.

- Indisponibilidade do servidor em caso dos mesmos saírem do ar

- A velocidade de processamento no que tange a taxa de transferência, desta maneira, se a internet não tiver uma boa banda, o sistema pode ser comprometido.

Apêndice G

Banco De Dados

Durante muito tempo os BDs foram construídos sobre sistemas de arquivos do sistema operacional para armazenar seus dados, desta maneira, a fim de poder controlar os dados bem como os usuários, a necessidade de um sistema de gestão surgiu naturalmente. Atualmente os BDs são operados pelos Sistemas Gerenciadores de Bancos de Dados - SGBD (em inglês, *Database Managment System*), que correspondem a uma coleção de programas que permitem aos usuários criarem e manipularem uma base de dados de maneira simples. Um SGBD é, assim, um sistema de *software* de propósito geral que facilita o processo de definir, construir e manipular bases de dados de diversas aplicações [10] [11].

- **Definir** uma base de dados envolve a especificação de tipos de dados a serem armazenados na base de dados.
- **Construir** uma base de dados é o processo de armazenar os dados em algum meio que seja controlado pelo SGBD.
- **Manipular** uma base de dados indica a utilização de funções como a de consulta, para recuperar dados específicos, modificação da base de dados para refletir mudanças no conteúdo do banco (inserções, atualizações e remoções), e geração de relatórios.

G.1 Modelos de Dados

Os SGBD podem ser diferenciados de acordo com a representação dos dados que são contidos no banco, sendo divididos em modelo hierárquico, em rede, relacional e orientado a objetos, detalhados a seguir [69] [70].

G.1.1 Modelo Hierárquico

Nesse modelo de dados, os dados são estruturados em hierarquias ou árvores. Os nós das hierarquias contêm ocorrências de registros, onde cada registro é uma coleção de campos (atributos), cada um contendo apenas uma informação. O registro da hierarquia que precede a outros é o registro-pai, os outros são chamados de registros-filhos.

O relacionamento entre um registro-pai e vários registros-filhos possui cardinalidade 1:N. Os dados organizados segundo este modelo podem ser acessados segundo uma sequência hierárquica com uma navegação do topo para as folhas e da esquerda para a direita.

A figura G.1 representa, simplificadamente, este modelo.

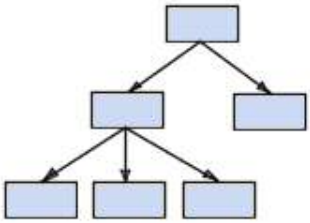

Figura G.1: Representação do Modelo Hierárquico

G.1.2 Modelo em Rede

O modelo em redes pode ser visto como uma extensão ao modelo hierárquico, porém eliminando-se o conceito de hierarquia e permitindo que um mesmo registro esteja envolvido em várias associações.

No modelo em rede, os registros são organizados em grafos onde aparece um único tipo de associação que define uma relação 1:N entre 2 tipos de registros: proprietário e membro. Desta maneira, dados dois relacionamentos 1:N entre os registros A e D e entre os registros C e D é possível construir um relacionamento M:N entre A e D.

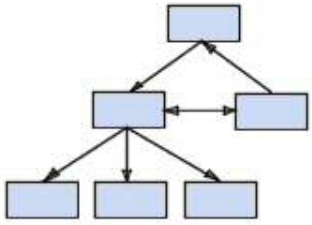

Figura G.2: Representação do Modelo em Rede

G.1.3 Modelo Relacional

O modelo relacional surgiu graças a necessidade de aumentar a independência de dados nos sistemas gerenciadores de banco de dados, prover um conjunto de funções apoiadas em álgebra relacional para armazenamento e recuperação de dados e permitir processamento *ad-hoc*.

O modelo relacional possui como base a teoria dos conjuntos e álgebra relacional, revelando-se como o modelo mais flexível e adequado ao solucionar os vários problemas que se colocam no nível da concepção e implementação da base de dados.

A estrutura fundamental do modelo relacional é a relação (tabela), no qual uma relação é constituída por um ou mais atributos (campos) que traduzem o tipo de dados a armazenar. Cada instância do esquema (linha) é chamada de tupla (registro).

O modelo relacional não tem caminhos pré-definidos para se fazer acesso aos dados como nos modelos que o precederam.

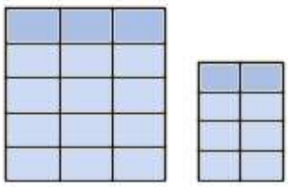

Figura G.3: Representação do Modelo Relacional

G.1.4 Modelo Orientado a Objetos

A motivação para o surgimento dos modelos orientados a objetos surgiu em função dos limites de armazenamento e representação semântica impostas no modelo relacional.

A habilidade para criar os tipos de dados necessários é uma característica das linguagens de programação orientadas a objetos. Contudo, estes sistemas necessitam guardar representações das estruturas de dados que utilizam no armazenamento permanente. A estrutura padrão para os bancos de dados orientados a objetos foi feita pelo *Object Database Management Group* (ODMG), que é formada por representantes dos principais fabricantes de banco de dados orientados a objeto disponíveis comercialmente.

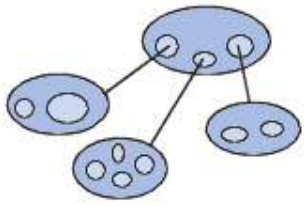

Figura G.4: Representação do Modelo Orientado a Objetos

G.2 Base de Dados x Processamento Tradicional de Arquivos

Como já apresentado, os BDs surgiram para substituir o armazenamento de dados em sistemas de arquivos do sistema operacional. A tabela G.1 [69] traz as principais características que diferem um sistema desenvolvido na perspectiva de banco de dados *versus* um desenvolvimento pelo tradicional gerenciamento de arquivos.

Processamento Tradicional de Arquivos	Base de Dados	Vantagem da Base de Dados
Definição dos dados é parte do código de programas de aplicação	Metadados	Eliminação de redundâncias
Dependência entre aplicação específica e dados	Capaz de permitir diversas aplicações	Eliminação de redundâncias
- - - -	Independência entre dados e programas	Facilidade de manutenção
Representação de dados ao nível físico	Representação conceitual através de dados e programas	Facilidade de manutenção
Cada visão é implementada por módulos específicos	Permite múltiplas visões	Facilidade de consultas

Tabela G.1: Comparação: Base de Dados x Processamento Tradicional de Arquivos

Apêndice H

Caracterização do *Pulse Sensor Amped*

De acordo com o fabricante, o *Pulse Sensor Amped* pode operar numa faixa entre 3v e 5v. A fim de determinar qual a melhor faixa de operação do mesmo, foi feito um estudo em laboratório de modo a caracterizar seu comportamento frente a diferentes tensões de alimentação. Este Apêndice visa mostrar os resultados desta análise.

H.1 Resposta do Sensor com Diferentes Tensões de Alimentação

O sensor foi submetido a diferentes tensões de alimentação, variando de 2,3 a 5v. O intuito era verificar em qual tensão sua resposta seria melhor.

Uma primeira análise foi feita através da alimentação do sensor por uma fonte de tensão DC, variando a tensão de entrada de 3 a 5v, e capturando a resposta do sensor num osciloscópio digital. As figuras H.1, H.2, H.3, H.4 e H.5 correspondem aos resultados obtidos.

Figura H.1: Resposta do Sensor Quando Alimentado com 3v

Figura H.2: Resposta do Sensor Quando Alimentado com 3,5v

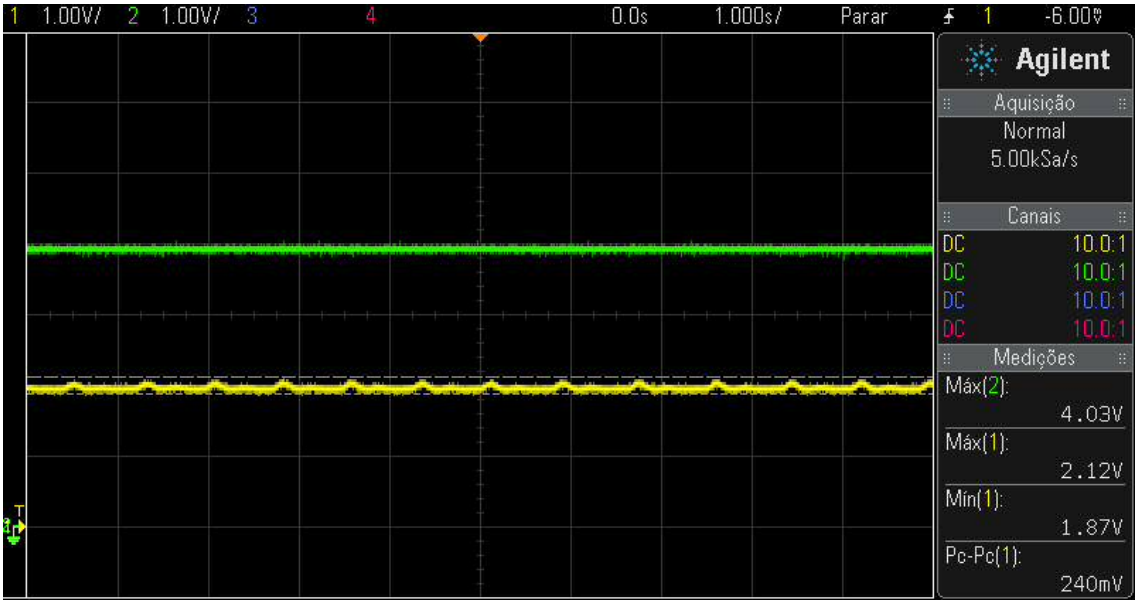

Figura H.3: Resposta do Sensor Quando Alimentado com 4v

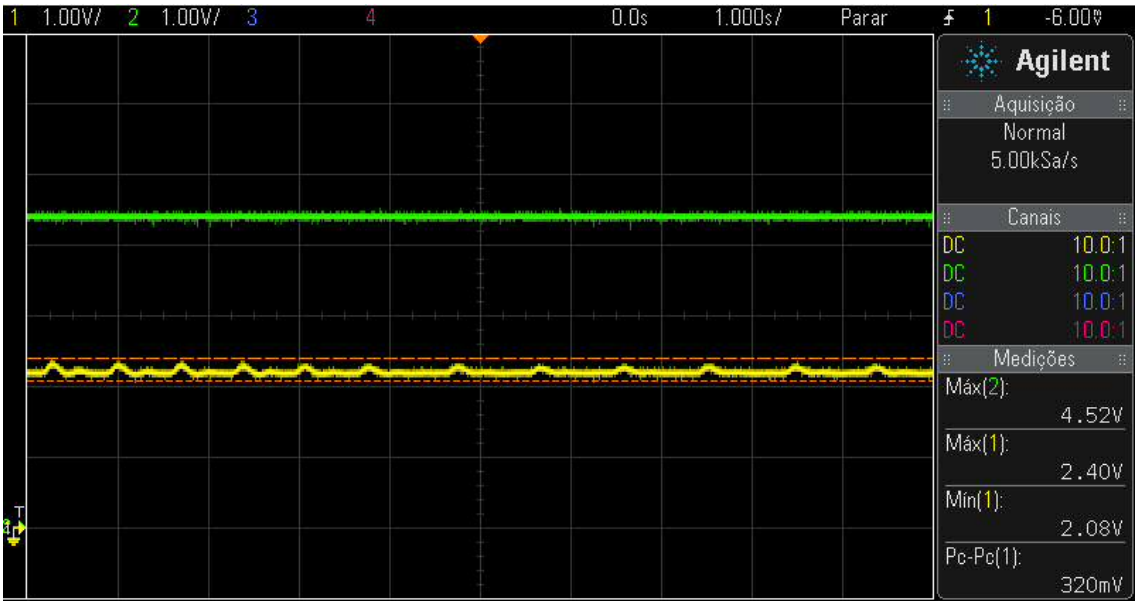

Figura H.4: Resposta do Sensor Quando Alimentado com 4,5v

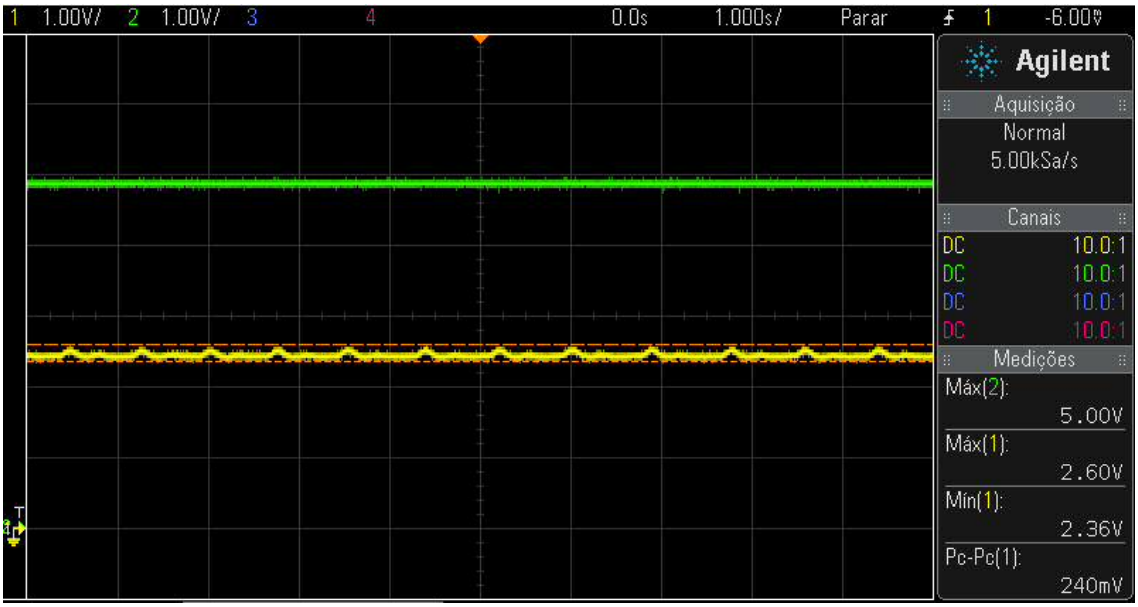

Figura H.5: Resposta do Sensor Quando Alimentado com 5v

Uma segunda análise foi feita variando a tensão em faixas fora de sua especificação de operação, para isto, alimentou-se o sensor com uma tensão logo abaixo de 3v (mais especificamente, 2,95v), 2,5v e procurou-se também o valor limiar de tensão para o qual o circuito parasse de captar sinal cardíaco (que ocorreu em 2,34v). Tais resultados são dispostos nas figuras H.6, H.7 e H.8.

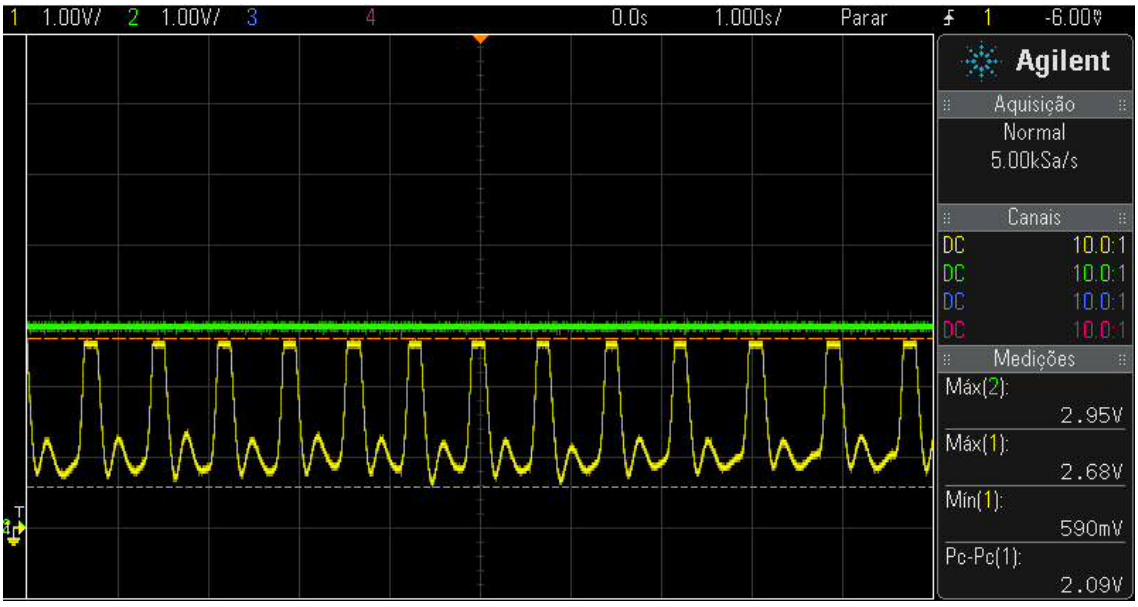

Figura H.6: Resposta do Sensor Quando Alimentado com 2,95v

Figura H.7: Resposta do Sensor Quando Alimentado com 2,5v

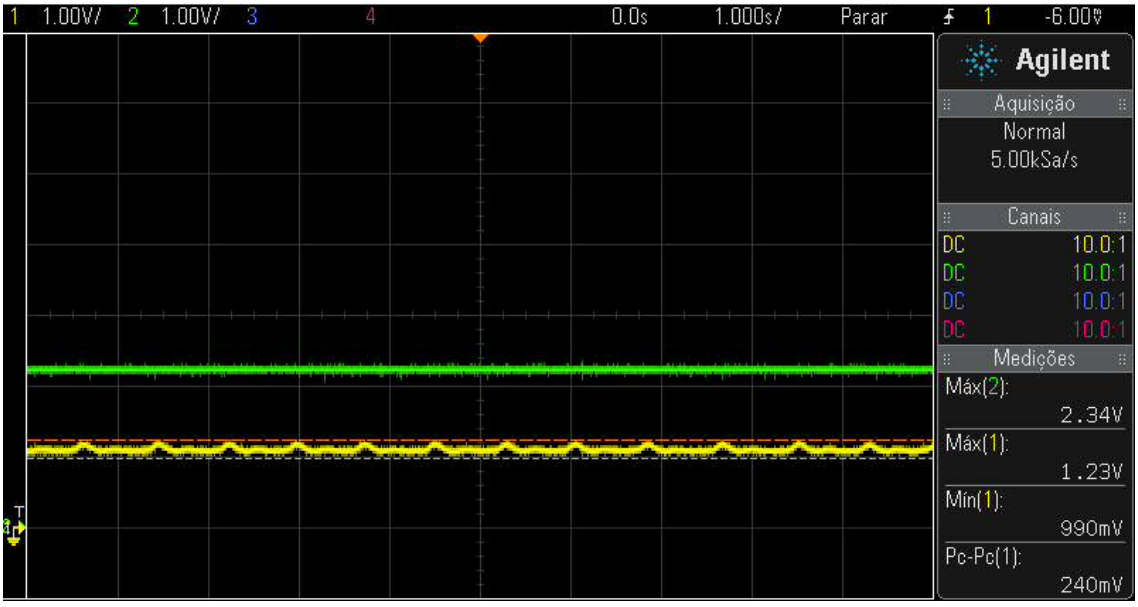

Figura H.8: Resposta do Sensor Quando Alimentado com 2,34v

H.2 Análise dos Resultados Obtidos

Com base nos resultados obtidos, foi elaborada a tabela H.1 e, com base na tabela, o gráfico da figura H.9.

Tensão de Alimentação	Tensão Máxima	Tensão Mínima	Tensão Pico a Pico
2.34	1.23	0.99	0.24
2.51	1.87	0.91	0.96
2.95	2.68	0.59	2.09
3.03	2.76	0.95	1.81
3.51	1.95	1.59	0.36
4.03	2.12	1.87	0.24
4.52	2.40	2.08	0.32
5.00	2.60	2.36	0.24

Tabela H.1: Resposta do Sensor com Diferentes Tensões de Alimentação

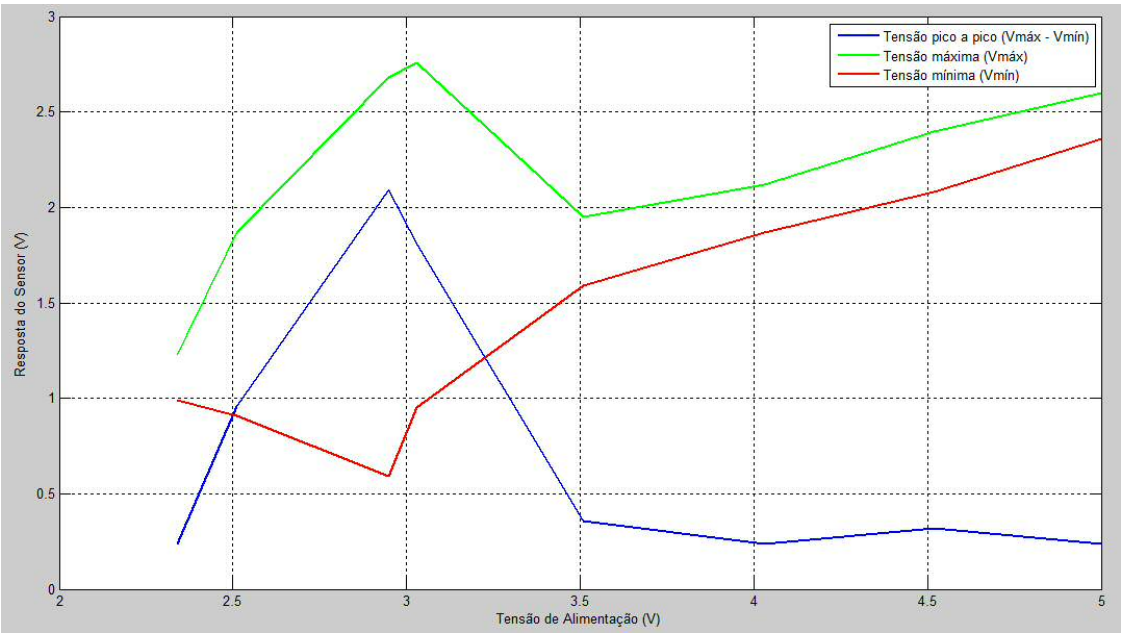

Figura H.9: Gráfico de Resposta do Sensor com Diferentes Tensões de Alimentação

Analisando s dados obtidos, percebe-se que o sensor possui uma melhor resposta, com maior resolução, em tensões enre 2,5 e 3,5 v. É interessante notar que a melhor resposta foi obtida para uma tensão logo abixo de 3v, contudo o sinal é ceifado e deformado. Dados tais resultados, a escolha para o desenvolvimento do projeto utilizando uma tensão de alimentação de 3v foi a mais pertinente.

Apêndice I

Código do Microprocessador

```
#include <18F2550.h>

// Diretivas de Funcionamento do PIC
// NOWDT: Watch Dog Timer desabilitado
// BROWNOUT: PIC reinicia quando ocorre Brown-out (alimentação cai repentinamente)
// NOLVP: Sem Low Voltage Programming
// INTRC: Uso de oscilador interno
#fuses NOWDT,BROWNOUT,NOLVP, INTRC
// Especificação de oscilação interna em 8 MHz
#use delay(internal=8M)
// Configuração serial: Baud rate de 9600, 8 bits e sem paridade
#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8,stream=PORT1)
#define LED5 PIN_B5
#define LED4 PIN_B4

void main()
{
   //Variáveis para cálculo do BPM
   unsigned int batimentos = 0;
   unsigned int bpm = 0;
   //Variáveis para envio do valor do BPM
   unsigned int centena = 0;
   unsigned int dezena = 0;
   unsigned int unidade = 0;
   // Código para controle de inicialização do programa
   output_high(LED5);
   delay_ms(3000);
   output_low(LED5);
```

```
29    // Configuração para Timer0 funcionar como contador
30    // RTCC_EXT_H_TO_L: Contagem será executada toda vez que sinal variar de alto
          nível para baixo
31    // RTCC_DIV_1: O resultado da contagem não será dividido por nenhum valor
32    setup_timer_0(RTCC_EXT_H_TO_L|RTCC_DIV_1);
33    while(1)
34    {
35       //Inicializa contador com valor 0
36       set_timer0(0);
37       //Acende/Apaga Led (controle de fluxo)
38       output_toggle(LED4);
39       // Realiza um atraso de 15 segundos
40       // Nesse tempo o contador conta os batimentos
41       delay_ms(15000);
42       // Atribui-se  a variavel bpmParcial o valor contado
43       batimentos = get_rtcc();
44       // batimentos: Número de batimentos cardíacos numa janela de 15 segundos
45       // bpm: Propagação de 'batimentos' para janela de 60 segundos
46       bpm = batimentos*4;
47       //Separa a unidade, dezena e centena do valor do BPM
48       centena = (int) bpm/100;
49       dezena = (int) (bpm - centena*100)/10;
50       unidade = (int) (bpm - (centena*100+dezena*10));
51       // Envia valor de 'bpm' para o módulo Bluetooth via transmissão serial
52       // Transmissão se incia com caracter de controle '*'
53       printf("*");
54       delay_ms(2000);
55       printf("%u", centena);
56       delay_ms(2000);
57       printf("%u", dezena);
58       delay_ms(2000);
59       printf("%u", unidade);
60       delay_ms(2000);
61    }
62 }
```

Apêndice J

Código de Configuração do Módulo HC-05

```
#include <18F2550.h>

// Diretivas de Funcionamento do PIC
// NOWDT: Watch Dog Timer desabilitado
// BROWNOUT: PIC reinicia quando ocorre Brown-out (alimentação cai repentinamente)
// NOLVP: Sem Low Voltage Programming
// INTRC: Uso de oscilador interno
#fuses NOWDT,BROWNOUT,NOLVP, INTRC
// Especificação de oscilação interna em 8 MHz
#use delay(internal=8M)
// Configuração serial: Baud rate de 38400, 8 bits e sem paridade
#use rs232(baud=38400,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8,stream=PORT1)
#define LED5 PIN_B5
#define LED4 PIN_B4

void main()
{
   //Configura PORT_B como saída e coloca os pinos em nível baixo
   set_tris_b(0x00);
    output_b(0x00);
   // Código para controle de inicialização do programa
   output_high(LED5);
   delay_ms(5000);
   output_low(LED5);
   //Reset do módulo
```

```
26      printf("AT+RESET\r\n");
27      output_high(LED4);
28      delay_ms(100);
29      //Configura Baud Rate do HC-05 em 9600, sem paridade, 1 stopbit
30      printf("AT+UART=9600,0,0\r\n");
31      output_low(LED4);
32      delay_ms(100);
33      //Configura nome do HC-05
34      printf("AT+NAME=PulseData\r\n");
35      output_high(LED4);
36      delay_ms(100);
37      //Configura senha do HC-05
38      printf("AT+PSWD=123456789\r\n");
39      output_low(LED4);
40      delay_ms(100);
41      while(1)
42      {
43          delay_ms(100);
44      }
45  }
```

Apêndice K

Código do Aplicativo para Android

K.1 Código da Main Activity

```
1  package com.example.pulsesensorapp;
2  import java.util.ArrayList;
3  import java.util.Calendar;
4  import java.util.Collection;
5  import java.util.Date;
6  import java.util.List;
7  import java.util.Timer;
8  import java.util.TimerTask;
9  import com.parse.FindCallback;
10 import com.parse.Parse;
11 import com.parse.ParseException;
12 import com.parse.ParseObject;
13 import com.parse.ParseQuery;
14 import android.app.Activity;
15 import android.content.Context;
16 import android.content.Intent;
17 import android.location.LocationManager;
18 import android.media.MediaPlayer;
19 import android.os.Bundle;
20 import android.os.PowerManager;
21 import android.util.Log;
22 import android.view.View;
23 import android.widget.Button;
24 import android.widget.EditText;
25 import android.widget.Toast;
26 import java.io.IOException;
```

```
27 import java.io.InputStream;
28 import java.lang.reflect.Method;
29 import java.util.UUID;
30 import org.json.JSONArray;
31 import android.bluetooth.BluetoothAdapter;
32 import android.bluetooth.BluetoothDevice;
33 import android.bluetooth.BluetoothSocket;
34 import android.os.Build;
35 import android.os.Handler;
36 import android.widget.TextView;
37
38 public class MainActivity extends Activity {
39    /* Usado para imprimir os logs no LogCat */
40    private static final String TAG = "Pulse Sensor App";
41    /* MediaPlayer é usada no controle do toque do alarme de emergência*/
42    MediaPlayer mediaPlayer;
43    /* LocationManager nos dá acesso aos serviços de localização geográfica do
          dispositivo, incluindo atualizações sobre coordenadas. */
44    LocationManager locationManager;
45    /* ParseObject é uma representação local de dados que podem ser salvos e
          recuperados da Parse cloud.*/
46    ParseObject parseUser;
47    /* Wake lock é um mecanismo que indica que a aplicação não pode ser desativada
          (tela fica acesa tempo todo, por exemplo) */
48    PowerManager.WakeLock wakeLock ;
49    /* Declaração de alguns componentes que compõem a User Interface */
50    Button logOff;
51    Button turnOff;
52    Button login;
53    EditText userCPF;
54    EditText password;
55    TextView txtArduino;
56    TextView textViewCPF;
57    TextView textViewSenha;
58    TextView textViewBatimentosCardiacos;
59    /* Handler te permite enviar e processar objetos mensagens associados com
          thread's. */
60    Handler h;
61    /* BluetoothAdapter é uma representação do adaptador bluetooth físico do
          dispositivo */
62    private BluetoothAdapter btAdapter = null;
63    /* Socket para comunicação bluetooth */
```

```
64    private BluetoothSocket btSocket = null;
65    /* Neste programa a StringBuilder funcionará como um buffer para os dados
          bluetooth que estão chegando no dispositivo */
66    private StringBuilder sb = new StringBuilder();
67    /* ConnectedThread funciona como um canal de entrada para estabelecimento da
          comexão */
68    private ConnectedThread mConnectedThread;
69    /* Variável utilizada para tratar os dados de entrada do bluetooth */
70    String[] pulseDataString = new String[10];
71    /* Indica o tipo de mensagem recebida pela comunicação bluetooth */
72    final int RECIEVE_MESSAGE = 1;
73    /* SPP UUID service */
74    private static final UUID MY_UUID = UUID.fromString("
          00001101-0000-1000-8000-00805F9B34FB");
75    /* MAC-address do módulo bluetooth - esse endereço é específico para o módulo
          que estamos usando neste trabalho */
76    private static String address = "20:13:02:22:00:22";
77    /* lastPulseSignal mantém um registo do último sinal de pulso medido */
78    String lastPulseSignal;
79    /* Uma lista/array dos mesmos dados que estão sendo salvos nos arrays do banco
          de dados */
80    Collection<String> GPSCollection = new ArrayList<String>();
81    Collection<Boolean> anomalyCollection = new ArrayList<Boolean>();
82    Collection<String> BPMCollection = new ArrayList<String>();
83    Collection<String> timeCollection = new ArrayList<String>();
84    /* Contadore utilizados no cálculo da média dos 10 últimos batimentos recebidos
           para que uma anomalia possa ser detectada */
85     int pulseDataCounter = 0;
86     Boolean stringComplete = false;
87     /* loginStatus nos diz se o usuário está logado ou não */
88     Boolean loginStatus = false;
89     /* anomaly nos diz se uma anomalia foi detectada ou não */
90     Boolean anomaly = false;
91     /* Contado de quantos em quantos registros nas arrays o banco de dados deve
           ser apagado. Ex: a cada 30 registros inseridos, apaga tudo e começa a
           escrever tudo de novo. */
92     int timerCounter = 0;
93     /* Controla o tempo que interrupções periódicas serão geradas para que haja
           escrita em banco */
94     MyTimerTask myTask = new MyTimerTask();
95     Timer myTimer = new Timer();
96     /* Classe especial para tratamento dos dados de GPS */
```

```
97      GPSTracker gps;
98      int contadorDados = -1;
99      int batimentoFinal = 65;
100     int batimentoParcial = 65;
101     /* Hora em que a última transmissão foi feita */
102     Date lastTransmissionTime;
103     /* Hora em que a última escrita em banco de dados foi feita */
104     Date lastDBTime;
105
106     @Override
107     protected void onCreate(Bundle savedInstanceState) {
108       super.onCreate(savedInstanceState);
109         setContentView(R.layout.activity_main);
110         turnOff = (Button) findViewById(R.id.turnOffButton);
111         login = (Button) findViewById(R.id.loginButton);
112       userCPF = (EditText)findViewById(R.id.userCPF);
113       password = (EditText)findViewById(R.id.userPassword);
114       txtArduino = (TextView)findViewById(R.id.textArduino);
115       textViewCPF = (TextView)findViewById(R.id.textView1);
116       textViewSenha = (TextView)findViewById(R.id.textView2);
117       textViewBatimentosCardiacos = (TextView)findViewById(R.id.textView3);
118
119       mediaPlayer = MediaPlayer.create(MainActivity.this, R.raw.alarm);
120       /* Faz com que o alarme toque apenas uma vez */
121        mediaPlayer.setLooping(false);
122        /* Configura a tela inicial que aparece após o usuário abrir a app */
123       txtArduino.setVisibility(View.INVISIBLE);
124       textViewBatimentosCardiacos.setVisibility(View.INVISIBLE);
125       /* Configura o controle da tela para ficar ativo o tempo todo, podendo porém
              diminuir o brilho em tempos de inatividade */
126       PowerManager pm = (PowerManager) getSystemService(Context.POWER_SERVICE);
127         wakeLock = pm.newWakeLock( PowerManager.SCREEN_DIM_WAKE_LOCK, "My wakelook
              ");
128         wakeLock.acquire();
129         gps = new GPSTracker(MainActivity.this);
130         /* Criação de uma nova classe e/ou um novo objeto no banco de dados */
131         Parse.initialize(this, "KW7yP7phAKov8cUVhoFuPQRE8JYy63uWEKInSTqu", "
              cIFY6FX2SA0YrQbrWxGvXYmz8PSWygNYCcQfJXuF");
132         /*
133         ParseObject parseUser = new ParseObject("PulseSensorClass");
134       parseUser.put("CPF", "123");
135       parseUser.put("password", "123");
```

```
136         parseUser.put("name", "Lara Bertini Vieira");
137         parseUser.put("userName", "Lara Bertini");
138         parseUser.put("bloodType", "O-");
139         parseUser.put("obs", "Nada de mais. Sou saudável mesmo.");
140         parseUser.put("status", false);
141         JSONArray BPMArray = new JSONArray();
142         JSONArray GPSArray = new JSONArray();
143         JSONArray anomalyArray = new JSONArray();
144         JSONArray timeArray = new JSONArray();
145         parseUser.put("BPM", BPMArray);
146         parseUser.put("GPS", GPSArray);
147         parseUser.put("anomaly", anomalyArray);
148         parseUser.put("time", timeArray);
149         parseUser.saveInBackground();
150           */
151           /* Uma maneira de se comunicar em android é através de "Messages" e "
                Handlers".
152            * O "Handler" recebe a mensagem e lida com ela. A mensagem é o conteúdo
                enviado.
153            * */
154           h = new Handler() {
155            public void handleMessage(android.os.Message msg) {
156               /* what: é um código definido pelo próprio usuário para que o
                    destinatário possa identificar o conteúdo da mensagem.
157                * Referencia: http://stackoverflow.com/questions/17642578/
                    obtainmessage-arguments-meaning
158                * */
159               switch (msg.what) {
160                   case RECIEVE_MESSAGE:
161                     byte[] readBuf = (byte[]) msg.obj;
162                     String strIncom = new String(readBuf, 0, msg.arg1);
163                     sb.append(strIncom);
164                     String stringBuffer = sb.substring(0, sb.length());
165                     String receivedString = stringBuffer.substring(stringBuffer.
                          length()-1, stringBuffer.length());
166                     Calendar calendar = Calendar.getInstance();
167                     lastTransmissionTime = calendar.getTime();
168                     /* Diminue o tamanho do buffer, para que o mesmo não cresça
                          infinitamente */
169                     if(sb.length() > 0){
170                        sb.setLength(sb.length() - 1);
171                     }
```

```
172                 if(receivedString.equals("*")){
173                     contadorDados = 0;
174                 }
175                 if(contadorDados==0 && !receivedString.equals("*")){
176                     batimentoParcial = Integer.valueOf(receivedString)*100;
177                     contadorDados++;
178                   }
179                 else if(contadorDados==1){
180                     batimentoParcial = batimentoParcial + Integer.valueOf(
                            receivedString)*10;
181                     contadorDados++;
182                   }
183                 else if(contadorDados==2){
184                     batimentoParcial = batimentoParcial + Integer.valueOf(
                            receivedString);
185                     batimentoFinal = batimentoParcial;
186                     contadorDados = 0;
187                        Log.d("Batimento Cardiaco", "******* Ultima Leitura: " +
                               batimentoFinal + " *******");
188                     /* A cada 10 pulsos recebidos, a média de batimentos
                            cardíacos é calculada */
189                        if(pulseDataCounter==10){
190                           pulseDataCounter = 0;
191                           stringComplete = true;
192                        }
193                        Integer integerBatimentoFinal = new Integer(batimentoFinal
                               );
194                        pulseDataString[pulseDataCounter] = integerBatimentoFinal.
                               toString();
195                        lastPulseSignal = pulseDataString[pulseDataCounter];
196                        txtArduino.setText(lastPulseSignal);
197                        pulseDataCounter ++;
198                        /* Se já temos pelo menos 10 amostras de pulso, podemos
                               começar a analisar os dados. O alarme será tocado e a
                               variável anomalia será verdadeira se alguma anomalia
                               for detectada. */
199                 if (stringComplete){
200                 int sum = 0;
201                 for (int i=0; i<10; i++){
202                    sum  = sum + Integer.valueOf(pulseDataString[i]);
203                 }
204                 int average = sum/10;
```

```
205                 double resultado = 0.00384*average*average - 0.357*average +
                        128;
206                 int resultado2 = (int) resultado;
207                 if((batimentoFinal > resultado2) && (loginStatus == true)){
208                    //Dispara alarme
209                    mediaPlayer.start();
210                    anomaly = true;
211                    }
212                 else if((batimentoFinal < 50) && (loginStatus == true)){
213                    //Dispara alarme
214                    mediaPlayer.start();
215                    anomaly = true;
216                 }
217                 }
218                    }
219                 break;
220             }
221             };
222     };
223       btAdapter = BluetoothAdapter.getDefaultAdapter();
224       checkBTState();
225   }
226
227   /*
228    * Ao criar uma atividade, o método onResume é automaticamente executado.
229    * Neste método ocorre a criação da thread que será responsável pela conexão
           com o módulo bluetooth
230    * */
231   @Override
232   public void onResume() {
233     super.onResume();
234     /* Configura um ponteiro para o dispositivo remoto usando o endereço do
           mesmo */
235     BluetoothDevice device = btAdapter.getRemoteDevice(address);
236     /* Duas coisas são necessárias para estabelecer uma conexão:
237      * Um MAC address
238      * O serviço ID ou UUID. Neste caso estamos usando UUID para SPP */
239   try {
240     btSocket = createBluetoothSocket(device);
241   } catch (IOException e) {
242     errorExit("Fatal Error", "In onResume() and socket create failed: " + e.
          getMessage() + ".");
```

```
243     }
244     /* A descoberta de novos dispositivos é um recurso de intensivo processamento.
245      * Temos que ter certeza que a descoberta não estará ativa enquanto tentamos
            conectar e passar alguma mensagem  */
246     btAdapter.cancelDiscovery();
247       /* Estabelecimento da conexão. Vai bloquear o computador até que a conexão
             seja realmente realizada */
248       Log.d(TAG, "...Connecting...");
249       try {
250         btSocket.connect();
251       } catch (IOException e) {
252         try {
253           btSocket.close();
254         } catch (IOException e2) {
255           errorExit("Fatal Error", "In onResume() and unable to close socket
                during connection failure" + e2.getMessage() + ".");
256         }
257       }
258       /* Cria um novo stream de dados para que possamos conversar com o servidor
             */
259       Log.d(TAG, "...Create Socket...");
260       mConnectedThread = new ConnectedThread(btSocket);
261       mConnectedThread.start();
262     }
263
264     @Override
265     public void onPause() {
266       super.onPause();
267       Log.d(TAG, "...In onPause()...");
268       try     {
269         btSocket.close();
270       } catch (IOException e2) {
271         errorExit("Fatal Error", "In onPause() and failed to close socket." + e2.
              getMessage() + ".");
272       }
273     }
274
275     /* Checa se tem suporte para bluetooth e se o mesmo está ligado */
276     private void checkBTState() {
277       /* Verifica se o dispositivo tem suporte para bluetooth. */
278       if(btAdapter==null) {
279         errorExit("Fatal Error", "Bluetooth not support");
```

```
280         } else {
281         /* Verifica se o bluetooth está ligado */
282           if (btAdapter.isEnabled()) {
283             Log.d(TAG, "...Bluetooth ON...");
284           } else {
285           /* Mostra uma tela ao usuário para que ele possa ligar o bluetooth */
286           Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE)
                  ;
287           startActivityForResult(enableBtIntent, 1);
288           }
289         }
290       }
291
292       private void errorExit(String title, String message){
293         Toast.makeText(getBaseContext(), title + " - " + message, Toast.LENGTH_LONG)
              .show();
294         finish();
295       }
296
297       public void startMethod(View v){
298         /* Encontra no bd o usuário com CPF e senha indicados */
299         ParseQuery<ParseObject> query = ParseQuery.getQuery("PulseSensorClass");
300         query.whereEqualTo("CPF", userCPF.getText().toString());
301         query.whereEqualTo("password", password.getText().toString());
302         query.findInBackground(new FindCallback<ParseObject>() {
303           @Override
304           public void done(List<ParseObject> userList, ParseException e) {
305                if (e == null) {
306                 /* Se existe apenas 1 usuário com o CPF e senha informados o login
                        foi bem sucedido */
307                   if(userList.size()==1){
308                     parseUser = (ParseObject) userList.get(0);
309                     Toast.makeText(MainActivity.this, "Login Successful!", Toast.
                          LENGTH_LONG).show();
310                     loginStatus = true;
311                     /* Dispara o timer para gerar interrupção de 1 em 1 minuto
                          para escrever no banco de dados */
312                     myTimer.schedule(myTask, 60000, 60000);
313                     /* Muda status do usuário para ativo */
314                     parseUser.put("status", true);
315                     /* Salva os dados no banco */
316                     parseUser.saveInBackground();
```

```
317                     Log.d("status depois do login", Boolean.toString(parseUser.
                            getBoolean("status")));
318                     /* Configura a tela principal da aplicação, onde aparece os
                            batimentos cardíacos */
319                     setMainScreen();
320                 }
321                 else if(userList.size()==0){
322                     Toast.makeText(MainActivity.this, "Try Again!", Toast.
                            LENGTH_LONG).show();
323                 }
324             } else {
325                 Log.d("score", "Error: " + e.getMessage());
326             }
327         }
328     });
329 }
330
331 public void turnOffMethod(View v){
332     /* Muda status do usuário para inativo, e salva no banco de dados */
333     parseUser.put("status", false);
334     parseUser.saveInBackground();
335     /* Muda o status de logado do usuário */
336     loginStatus = false;
337     /* Libera o wakelock - agora a app já pode ficar inativa */
338     wakeLock.release();
339     /* Libera os recursos necessários para tocar a música */
340     mediaPlayer.release();
341     /* Fecha a aplicação */
342     onPause();
343     finish();
344 }
345
346 /* Configura os elementos gráficos da tela principal */
347 public void setMainScreen(){
348     userCPF.setVisibility(View.INVISIBLE);
349     password.setVisibility(View.INVISIBLE);
350     login.setVisibility(View.INVISIBLE);
351     txtArduino.setVisibility(View.VISIBLE);
352     textViewCPF.setVisibility(View.INVISIBLE);
353     textViewSenha.setVisibility(View.INVISIBLE);
354     textViewBatimentosCardiacos.setVisibility(View.VISIBLE);
355 }
```

```
356
357     /* Cria o socket responsável pela comunicação bluetooth */
358     private BluetoothSocket createBluetoothSocket(BluetoothDevice device) throws
            IOException {
359         if(Build.VERSION.SDK_INT >= 10){
360             try {
361                 final Method  m = device.getClass().getMethod("
                        createInsecureRfcommSocketToServiceRecord", new Class[] { UUID
                        .class });
362                 return (BluetoothSocket) m.invoke(device, MY_UUID);
363             } catch (Exception e) {
364                 Log.e(TAG, "Could not create Insecure RFComm Connection",e);
365             }
366         }
367         return  device.createRfcommSocketToServiceRecord(MY_UUID);
368     }
369 }
```

K.2 Código da Classe ConnectedThread

```
1 /* Gera thread para rodar em pararelo com a UI thread a fim de tratar a entrada de
       dados por bluetooth.
2  * Essa thread ficará num loop infinito ouvindo o canal de entrada de dados. */
3 private class ConnectedThread extends Thread {
4     /* The goal of InputStream and OutputStream is to abstract different ways to
          input and output:
5      * whether the stream is a file, a web page, or the screen shouldn't matter.
6      * All that matters is that you receive information from the stream (or send
          information into that stream.)
7      * Referência: http://stackoverflow.com/questions/1830698/what-is-inputstream-
          output-stream-why-do-we-use-them-and-when-do-we-use-each
8      * */
9     /* Cria canal de comunicação para chegada de dados */
10    private final InputStream mmInStream;
11    public ConnectedThread(BluetoothSocket socket) {
12        InputStream tmpIn = null;
13        try {
14            tmpIn = socket.getInputStream();
15        } catch (IOException e) { }
16        mmInStream = tmpIn;
17    }
18    /* Assim que a thread foi criada, o método run é executado */
```

```
19    public void run() {
20       /* Cria um buffer para armazenar os dados que chegam */
21       //byte[] buffer = new byte[4*1024];
22       byte[] buffer = new byte[10*1024];
23       int bytes;
24       /* Começa e permanece num loop infinito ouvindo o canal de entrada. */
25       Log.d(TAG, "...Comeca a ouvir InStream...");
26       while (true) {
27          try {
28             /* Faz a leitura do canal de entrada e armazena os dados lidos no
                   buffer */
29             bytes = mmInStream.read(buffer);
30             /* Envia a mensagem para que o handler a possa tratar */
31             h.obtainMessage(RECIEVE_MESSAGE, bytes, -1, buffer).sendToTarget();
                      // Send to message queue Handler
32          } catch (IOException e) {
33             Log.d(TAG, "...Nao deu certo...");
34             break;
35          }
36       }
37    }
38 }
```

K.3 Código da Classe MyTimerTask

```
1 /* Essa classe foi gerada para que interrupções periódicas ocorram para que dados
     sejam armazenados no bd Parse. */
2     class MyTimerTask extends TimerTask {
3       public void run() {
4       /* Caso já houver o número limite de registros nas arrays armazenadas no
            banco de dados, todos os dados serão removidos e a inserção de novos
            dados acontecerá periodicamente. */
5       if(timerCounter==3){
6          /* Deleta dados das arrays armazenadas no banco de dados */
7          parseUser.removeAll("GPS", GPSCollection);
8          GPSCollection.clear();
9          parseUser.removeAll("anomaly", anomalyCollection);
10         anomalyCollection.clear();
11         parseUser.removeAll("BPM", BPMCollection);
12         BPMCollection.clear();
13         parseUser.removeAll("time", timeCollection);
14         timeCollection.clear();
```

```
15          /* Salva as alterações efetuadas. */
16          parseUser.saveInBackground();
17      }
18
19        /* Inícia-se a adição de dados no bd */
20      /* *** Dados GPS *** */
21      /* Checa se o dispositivo tem compatibilidade com bluetooth */
22        if(gps.canGetLocation()){
23            double latitude = gps.getLatitude();
24            double longitude = gps.getLongitude();
25            parseUser.add("GPS", "lat:" + latitude + "x" + "long:" + longitude);
26            GPSCollection.add("lat:" + latitude + "x" + "long:" + longitude);
27            Log.d("LOCALIZACAO GPS", "Your Location is - \nLat: " + latitude + "\
                  nLong: " + longitude);
28        }else{
29          /* Caso haja problemas na recepção dos dados GPS, pedir para usuário
                ativa GPS ou internet */
30            gps.showSettingsAlert();
31        }
32        /* *** Dados sobre a ocorrência de anomalia *** */
33        parseUser.add("anomaly", anomaly);
34        anomalyCollection.add( anomaly);
35        /* Para que uma nova anomalia possa ser detectada o valor de anomalia é
              configurado para falso novamente */
36        anomaly = false;
37        /* *** Dados BPM *** */
38        /* Atualiza dados do último batimento cadíaco registrado */
39        parseUser.add("BPM", lastPulseSignal);
40        BPMCollection.add(lastPulseSignal);
41        /* *** Dados de calendário *** */
42        Calendar calendar = Calendar.getInstance();
43        parseUser.add("time", Integer.toString(calendar.get(Calendar.HOUR_OF_DAY))
              + "h" + Integer.toString(calendar.get(Calendar.MINUTE)) + "min");
44        timeCollection.add(Integer.toString(calendar.get(Calendar.HOUR_OF_DAY))+ "
              h" + Integer.toString(calendar.get(Calendar.MINUTE)) + "min");
45        Log.d("HORAS", Integer.toString(calendar.get(Calendar.HOUR_OF_DAY))+ "h" +
               Integer.toString(calendar.get(Calendar.MINUTE)) + "min");
46        /* Verifica se o último dado recebido foi a menos de 1 min. Se não tiver
              sido acusa que o dispositivo foi danificado e dispara o alarme */
47      lastDBTime = calendar.getTime();
48       long diffMinutes = (lastDBTime.getTime() - lastTransmissionTime.getTime())
             / (60 * 1000);
```

```
49          if (diffMinutes > 1){
50            mediaPlayer.start();
51              anomaly = true;
52          }
53           /* Salva as útimas modificações no bd */
54           parseUser.saveInBackground();
55           timerCounter ++;
56       }
57     }
```

K.4 Código da Classe GPSTracker

```
1  package com.example.pulsesensorapp;
2  import android.app.AlertDialog;
3  import android.app.Service;
4  import android.content.Context;
5  import android.content.DialogInterface;
6  import android.content.Intent;
7  import android.location.Location;
8  import android.location.LocationListener;
9  import android.location.LocationManager;
10 import android.os.Bundle;
11 import android.os.IBinder;
12 import android.provider.Settings;
13 import android.util.Log;
14
15 public class GPSTracker extends Service implements LocationListener{
16    private final Context mContext;
17     /* Flag de status do GPS */
18     boolean isGPSEnabled = false;
19     /* Flag de status da rede */
20     boolean isNetworkEnabled = false;
21     /* Flag de status de comunicação com o GPS */
22     boolean canGetLocation = false;
23     /* Localizaçao */
24     Location location;
25     /* Latitude */
26     double latitude;
27     /* Longitude */
28     double longitude;
29     /* Distância mínima em metros necessária para atualizar a localização GPS.
          Neste caso estamos considerando 10 metros. */
```

```
30      private static final long MIN_DISTANCE_CHANGE_FOR_UPDATES = 10;
31      /* Tempo mínimo entre atualizações de dados GPS. Neste caso usaremos 1 minuto.
            */
32      private static final long MIN_TIME_BW_UPDATES = 1000 * 60 * 1;
33      /* LocationManager faz com que possamos ter acesso aos serviços de localização
            do dispositivo. */
34      protected LocationManager locationManager;
35      public GPSTracker(Context context) {
36          this.mContext = context;
37          getLocation();
38      }
39      public Location getLocation() {
40          try {
41              locationManager = (LocationManager) mContext.getSystemService(
                    LOCATION_SERVICE);
42              /* Verifica o status do GPS */
43              isGPSEnabled = locationManager.isProviderEnabled(LocationManager.
                    GPS_PROVIDER);
44              /* Verifica o status da rede */
45              isNetworkEnabled = locationManager.isProviderEnabled(LocationManager.
                    NETWORK_PROVIDER);
46              /* Nenhum provedor para localização está ativado */
47              if (!isGPSEnabled && !isNetworkEnabled) {
48                 showSettingsAlert();
49              } else {
50                  this.canGetLocation = true;
51                  /* Se a rede estiver disponível */
52                  if (isNetworkEnabled) {
53                    /* Atualiza os dados sobre localização */
54                      locationManager.requestLocationUpdates(LocationManager.
                            NETWORK_PROVIDER, MIN_TIME_BW_UPDATES,
                            MIN_DISTANCE_CHANGE_FOR_UPDATES, this);
55                      Log.d("Network", "Network");
56                      if (locationManager != null) {
57                          location = locationManager.getLastKnownLocation(
                                LocationManager.NETWORK_PROVIDER);
58                          if (location != null) {
59                              latitude = location.getLatitude();
60                              longitude = location.getLongitude();
61                          }
62                      }
63                  }
```

```
64                  /* Se o GPS estiver ativo, pegar os dados sobre latitude e
                        longitude usando os serviços de GPS */
65                  if (isGPSEnabled) {
66                      if (location == null) {
67                          locationManager.requestLocationUpdates(LocationManager.
                                GPS_PROVIDER, MIN_TIME_BW_UPDATES,
                                MIN_DISTANCE_CHANGE_FOR_UPDATES, this);
68                          Log.d("GPS Enabled", "GPS Enabled");
69                          if (locationManager != null) {
70                              location = locationManager.getLastKnownLocation(
                                    LocationManager.GPS_PROVIDER);
71                              if (location != null) {
72                                  latitude = location.getLatitude();
73                                  longitude = location.getLongitude();
74                              }
75                          }
76                      }
77                  }
78              }
79          } catch (Exception e) {
80              e.printStackTrace();
81          }
82          return location;
83      }
84      /* Pára de usar os serviços de GPS na aplicação */
85      public void stopUsingGPS(){
86          if(locationManager != null){
87              locationManager.removeUpdates(GPSTracker.this);
88          }
89      }
90      /* Função retorna a latitude do dispositivo */
91      public double getLatitude(){
92          if(location != null){
93              latitude = location.getLatitude();
94          }
95          return latitude;
96      }
97      /* Função retorna a longitude do dispositivo */
98      public double getLongitude(){
99          if(location != null){
100             longitude = location.getLongitude();
101         }
```

```
102         return longitude;
103     }
104     /* Checa se o GPS ou wifi estão ativos */
105     public boolean canGetLocation() {
106         return this.canGetLocation;
107     }
108     /* Verifica se o GPS está disponível para uso */
109     public boolean isGPSEnabled() {
110         return this.isGPSEnabled;
111     }
112     /* Verifica se a rede está disponível para uso */
113     public boolean isNetworkEnabled() {
114         return this.isNetworkEnabled;
115     }
116     /* Caso o GPS ou rede não estejam ativos, mostra a o menu Settings Options
            para que o usuário possa ativá-los */
117     public void showSettingsAlert(){
118         AlertDialog.Builder alertDialog = new AlertDialog.Builder(mContext);
119         /* Configura o título da caixa de diálogo que aparecerá */
120         alertDialog.setTitle("GPS in settings");
121         /* Configura a mensagem que aparecerá na caixa de diálogo */
122         alertDialog.setMessage("GPS is not enabled. Do you want to go to settings
                menu?");
123         /* Ação ao pressionar o botão settings */
124         alertDialog.setPositiveButton("Settings", new DialogInterface.
                OnClickListener() {
125             public void onClick(DialogInterface dialog,int which) {
126                 Intent intent = new Intent(Settings.
                        ACTION_LOCATION_SOURCE_SETTINGS);
127                 mContext.startActivity(intent);
128             }
129         });
130         /* Ao pressionar o botão cancelar */
131         alertDialog.setNegativeButton("Cancel", new DialogInterface.
                OnClickListener() {
132             public void onClick(DialogInterface dialog, int which) {
133             dialog.cancel();
134             }
135         });
136         /* Mostra o diálogo de alerta */
137         alertDialog.show();
138     }
```

```
139     @Override
140     public void onLocationChanged(Location location) {}
141     @Override
142     public void onProviderDisabled(String provider) {}
143     @Override
144     public void onProviderEnabled(String provider) {}
145     @Override
146     public void onStatusChanged(String provider, int status, Bundle extras) {}
147     @Override
148     public IBinder onBind(Intent intent) {return null;}
149 }
```

K.5 Código do Layout

```
1  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:paddingBottom="@dimen/activity_vertical_margin"
6      android:paddingLeft="@dimen/activity_horizontal_margin"
7      android:paddingRight="@dimen/activity_horizontal_margin"
8      android:paddingTop="@dimen/activity_vertical_margin"
9      tools:context=".MainActivity" >
10     <EditText
11         android:id="@+id/userCPF"
12         android:layout_width="wrap_content"
13         android:layout_height="wrap_content"
14         android:layout_alignParentLeft="true"
15         android:layout_alignParentTop="true"
16         android:layout_marginLeft="67dp"
17         android:layout_marginTop="65dp"
18         android:ems="10" >
19         <requestFocus />
20     </EditText>
21     <EditText
22         android:id="@+id/userPassword"
23         android:layout_width="wrap_content"
24         android:layout_height="wrap_content"
25         android:layout_alignLeft="@+id/userCPF"
26         android:layout_below="@+id/userCPF"
27         android:layout_marginTop="22dp"
28         android:ems="10"
```

```
29          android:inputType="textPassword" />
30      <Button
31          android:id="@+id/loginButton"
32          android:layout_width="fill_parent"
33          android:layout_height="wrap_content"
34          android:layout_centerHorizontal="true"
35          android:layout_centerVertical="true"
36          android:onClick="startMethod"
37          android:text="Login" />
38      <Button
39          android:id="@+id/turnOffButton"
40          android:layout_width="wrap_content"
41          android:layout_height="wrap_content"
42          android:layout_alignParentBottom="true"
43          android:layout_alignRight="@+id/userCPF"
44          android:layout_marginBottom="19dp"
45          android:onClick="turnOffMethod"
46          android:text="Off" />
47      <TextView
48          android:id="@+id/textView1"
49          android:layout_width="wrap_content"
50          android:layout_height="wrap_content"
51          android:layout_alignBottom="@+id/userCPF"
52          android:layout_alignParentLeft="true"
53          android:text="CPF:" />
54      <TextView
55          android:id="@+id/textView2"
56          android:layout_width="wrap_content"
57          android:layout_height="wrap_content"
58          android:layout_alignBottom="@+id/userPassword"
59          android:layout_alignParentLeft="true"
60          android:text="Senha:" />
61      <TextView
62          android:id="@+id/textArduino"
63          android:layout_width="wrap_content"
64          android:layout_height="wrap_content"
65          android:layout_below="@+id/loginButton"
66          android:layout_centerHorizontal="true"
67          android:layout_marginTop="39dp"
68          android:text="____" />
69      <TextView
70          android:id="@+id/textView3"
```

```
71          android:layout_width="wrap_content"
72          android:layout_height="wrap_content"
73          android:layout_above="@+id/turnOffButton"
74          android:layout_centerHorizontal="true"
75          android:layout_marginBottom="24dp"
76          android:text="BPM" />
77 </RelativeLayout>
```

K.6 Código do Manifest

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="com.example.pulsesensorapp"
4     android:versionCode="1"
5     android:versionName="1.0" >
6     <uses-sdk
7         android:minSdkVersion="8"
8         android:targetSdkVersion="18" />
9     <uses-permission android:name="android.permission.INTERNET" />
10    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
11    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" >
12    </uses-permission>
13    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" >
14    </uses-permission>
15    <uses-permission android:name="android.permission.BLUETOOTH" />
16    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
17    <uses-permission  android:name="android.permission.WAKE_LOCK"  />
18    <application
19        android:allowBackup="true"
20        android:icon="@drawable/ic_launcher"
21        android:label="@string/app_name"
22        android:theme="@style/AppTheme" >
23        <activity
24            android:name="com.example.pulsesensorapp.MainActivity"
25            android:label="@string/app_name" >
26            <intent-filter>
27                <action android:name="android.intent.action.MAIN" />
28                <category android:name="android.intent.category.LAUNCHER" />
29            </intent-filter>
30        </activity>
31        <activity
32            android:name="com.example.pulsesensorapp.LiveData"
```

```
33              android:label="@string/title_activity_live_data" >
34          </activity>
35          <activity
36              android:name="com.example.pulsesensorapp.LocationActivity"
37              android:label="@string/title_activity_location" >
38          </activity>
39          <activity
40              android:name="com.example.pulsesensorapp.BluetoothActivity"
41              android:label="@string/title_activity_bluetooth" >
42          </activity>
43      </application>
44  </manifest>
```