

UNIVERSIDADE DO ESTADO DO AMAZONAS - UEA
ESCOLA SUPERIOR DE TECNOLOGIA
ENGENHARIA DE COMPUTAÇÃO

LEANDRO MAURÍCIO ARAÚJO BENTES

**SISTEMA DE SEGURANÇA VEICULAR COM
USO DE GPS**

Manaus

2013

LEANDRO MAURÍCIO ARAÚJO BENTES

SISTEMA DE SEGURANÇA VEICULAR COM USO DE GPS

Trabalho de Conclusão de Curso apresentado
à banca avaliadora do Curso de Engenharia
de Computação, da Escola Superior de
Tecnologia, da Universidade do Estado do
Amazonas, como pré-requisito para obtenção
do título de Engenheiro de Computação.

Orientador: Prof. Dr. Raimundo Corrêa de Oliveira

Manaus

2013

***Universidade do Estado do Amazonas - UEA
Escola Superior de Tecnologia - EST***

Reitor:

Cleinaldo de Almeida Costa

Vice-Reitor:

Raimundo de Jesus Teixeira Barradas

Diretor da Escola Superior de Tecnologia:

Cleto Cavalcante de Souza Leal

Coordenador do Curso de Engenharia de Computação:

Raimundo Corrêa de Oliveira

Coordenador da Disciplina Projeto Final:

Tiago Eugenio de Melo

Banca Avaliadora composta por:

Data da Defesa: / /2013.

Prof. Dr. Raimundo Corrêa de Oliveira (Orientador)

Prof. Dr. Jucimar Maia da Silva Júnior

Prof. Dr. Ricardo da Silva Barboza

CIP - Catalogação na Publicação

B475s BENTES, Leandro

Sistema de Segurança Veicular com uso de GPS / Leandro Bentes; [orientado por] Prof. Dr. Raimundo Corrêa de Oliveira - Manaus: UEA, 2013.

240 p.: il.; 30cm

Inclui Bibliografia

Trabalho de Conclusão de Curso (Graduação em Engenharia de Computação). Universidade do Estado do Amazonas, 2013.

CDU: 004.78

LEANDRO MAURÍCIO ARAÚJO BENTES

SISTEMA DE SEGURANÇA VEICULAR COM USO DE GPS

Trabalho de Conclusão de Curso apresentado à banca avaliadora do Curso de Engenharia de Computação, da Escola Superior de Tecnologia, da Universidade do Estado do Amazonas, como pré-requisito para obtenção do título de Engenheiro de Computação.

Aprovado em: / /2013

BANCA EXAMINADORA

Prof. Raimundo Corrêa de Oliveira, Doutor
UNIVERSIDADE DO ESTADO DO AMAZONAS

Prof. Jucimar Maia da Silva Júnior, Doutor
UNIVERSIDADE DO ESTADO DO AMAZONAS

Prof. Ricardo da Silva Barboza, Doutor
CENTRO UNIVERSITÁRIO DO NORTE

Agradecimentos

Ao finalizar este trabalho após uma caminhada de cinco anos, tive a oportunidade de conviver com companheiros de turma singulares, que se tornaram amigos para todas as horas, além de podem contar com professores que amam multiplicar conhecimento e acima de tudo, sua experiência de vida. Agradeço a todos estes pelo apoio e também aqueles que trilharam comigo este caminho:

Minha família, que ficou diversas vezes sem minha companhia em confraternizações e sempre indagava pela minha presença, mas sempre incentivou e apoiou minhas escolhas. À minha namorada que desde o começo compreendeu a abdicação do meu tempo com ela em prol dos estudos.

Aos amigos que iniciaram comigo esta caminhada no mundo das exatas, se aventurando nas competições de robótica com sede de conhecimento e adrenalina com, mesmo que tenhamos trilhado diferentes caminhos agradeço por poder dividir alegrias e angústias durante o tempo em que ficamos em na faculdade.

Resumo

A preocupação com a segurança dos próprios bens tem se tornado cada vez mais intensa, uma vez que atos onde criminosos praticam roubos ou furtos ocorrem com bastante frequência. No caso de veículos automotores, o proprietário fica sem poder algum para evitar o crime, já que uma reação pode colocar em risco sua vida e de outros que o acompanham. Nesta situação se faz necessário o uso de algum mecanismo que permita ao usuário recuperar sua propriedade minimizando os riscos à sua integridade e de outros que o estejam acompanhando no momento da ação criminosa.

Este trabalho apresenta o desenvolvimento de um sistema voltado a segurança veicular com uso de GPS, o que permite o rastreamento, em tempo real, do automóvel por meio de uma aplicação Web. Além da função de rastreamento, o sistema permite realizar o bloqueio, ou seja, é possível desligar o automóvel remotamente e de modo irrevogável, impedindo que este sofra danos causados pela condução agressiva de meliantes em tentativas de fuga ou mesmo na realização de novos crimes. A combinação da localização geográfica com o desligamento remoto reduz as chances de danos ao patrimônio e evita que o proprietário tente reagir no momento de uma abordagem, pois poderá contar com auxílio do sistema.

A solução se baseia em dois módulos principais: um sistema de propósito específico embarcado no automóvel e um serviço Web responsável pela recepção de informações, transmissão de comandos bloqueio/desbloqueio e interface com o usuário. O processo de desenvolvimento utilizou princípios de Engenharia de Software aplicada a Sistemas Embarcados, permitindo mapear de forma sucinta os requisitos e a arquitetura inicial. O uso da modelagem UML no projeto do módulo embarcado auxiliou na definição dos componentes de hardware a serem utilizados e no método de comunicação com o serviço Web. Ao final da implementação foram realizados testes de campo, além de simulações de carga no serviço Web para detectar e corrigir possíveis gargalos.

Palavras Chave: GPS, GSM, Webservice, Sistema Embarcado

Abstract

The concern for the safety of our property has become increasingly intense since acts which criminals practice robberies or thefts occur quite often. In the case of motor vehicles, the owner is powerless to prevent the crime, since a reaction can endanger your life and others that accompany it. In this situation, it is necessary to use some mechanism that allows the user to recover their property while minimizing the risks to your health and others.

This paper presents the development of a facing vehicle safety system using GPS which allows tracking in real time of the car through a web application. Besides tracking function , the system allows the blocking, or is, you can remotely and irrevocably turn off the car, preventing it from being damaged caused by aggressive driving of miscreants in escape attempts or even make new crimes. The combination of geographic location with the remote shutdown reduces the chances of damage to property and prevents the owner try to respond during a criminal act because it can rely on the aid system.

The solution is based on two main modules : a system of special purpose embedded in the car and a Web service responsible for receiving information, transmitting commands of lock/unlock and user interface. The development process used software engineering principles applied to embedded systems allowing mapping succinctly initial requirements and architecture. The use of UML in the design of embedded module assisted in defining the hardware components to be used and the method of communication with the Web service implementation. At the end of the field tests were performed and load simulations in Web service to detect and correct potential bottlenecks.

Keywords : GPS, GSM, Webservice, Embedded System

Sumário

Lista de Tabelas	ix
Lista de Figuras	x
Lista de Códigos	xi
1 Introdução	1
1.1 Descrição do Problema	1
1.2 Trabalhos Relacionados	2
1.3 Contribuições	3
1.4 Metodologia	4
1.5 Organização	5
2 Fundamentação Teórica	6
2.1 Sistema Embarcado	6
2.2 Projeto de Sistemas Embarcados	8
2.3 Microcontroladores	9
2.4 Plataforma Arduino	10
2.4.1 Arduino Shields	12
2.5 Placa Arduino Uno	13
2.5.1 Especificações Arduino Uno	14
2.5.2 Alimentação da Placa Arduino Uno	14
2.5.3 Entrada e Saída	16
2.5.4 Programação	17
2.6 Tecnologia GPS	18
2.6.1 Modelo de Posicionamento GPS	19
2.7 Arduino GPS Shield Kit	20

3 Desenvolvimento do Protótipo	23
3.1 Modelo Conceitual	23
3.2 Especificação de Requisitos	24
3.3 Arquitetura do Sistema	24
3.4 Módulo Veicular Embarcado - Modelagem	24
3.5 Módulo Veicular Embarcado - Hardware	25
3.6 Módulo Web (WebManager) - Introdução	30
3.7 Módulo Web (WebManager) - Modelagem	31
3.8 Módulo Web (WebManager) - Desenvolvimento	33
4 Resultados e Discussões	35
4.1 Módulo Web (WebManager) - Testes de Carga	35
4.2 Resultados	36
4.3 Próximos Passos	38
4.4 Algoritmos	38

Lista de Tabelas

2.1	Modelos de placa Arduino	11
2.2	Resumo das Especificações - Arduino Uno	14
2.3	Especificações operacionais do módulo GPS EM-406A	22
2.4	Descrições das seções da mensagem NMEA 0183	22

Listas de Figuras

2.1	Um roteador wireless é um sistema embarcado	7
2.2	Microcontrolador Atmel	10
2.3	Placa Arduino com shields conectados	13
2.4	Placa Arduino Uno em detalhes	14
2.5	Fonte de alimentação para testes em bancada	15
2.6	IDE Arduino	17
2.7	Modelo conceitual da constelação de satélites GPS	18
2.8	Conceito para cálculo da coordenada de um ponto via GPS	19
2.9	Elementos do sistema GPS	20
2.10	Kit GPS	20
2.11	Esquema elétrico do GPS Shield	21
2.12	Mensagem NMEA 0183	22
3.1	Visão de operação do sistema e os elementos envolvidos	23
3.2	Casos de uso do sistema	24
3.3	Arquitetura do Sistema	25
3.4	Classes do Módulo Veicular	25
3.5	Placa Arduino Uno	26
3.6	GPS Shield e Módulo EM 406	27
3.7	Cellular Shield com SM5100B	27
3.8	Classe de comunicação com SM5100B	28
3.9	Módulo de I/O com veículo	29
3.10	Módulo veicular montado	29
3.11	Requisição HTTP feita pelo módulo veicular	30
3.12	Modelo de classes do WebManager	32
3.13	Model - Entidades do sistema	33
3.14	Tela de Login do Sistema	34

3.15 Tela de Rastreamento	34
4.1 Gráfico do consumo de memória heap	36
4.2 Gráfico do uso de CPU	36
4.3 Integração do protótipo	37
4.4 Dados coletados	38

Lista de Códigos

4.4.1 <i>Código do Módulo Embarcado Parte 1</i>	39
4.4.2 <i>Código do Módulo Embarcado Parte 2</i>	40
4.4.3 <i>Código do Módulo Embarcado Parte 3</i>	41
4.4.4 <i>Código do Módulo Embarcado Parte 4</i>	42

Capítulo 1

Introdução

1.1 Descrição do Problema

A quantidade de roubos e furtos de veículos no Brasil vem avançando de forma significativa durante os últimos anos, aumentando proporcionalmente ao número de automóveis em circulação. Informações das secretarias estaduais de segurança pública do Detran e do Denatran, mostram que as cidades do Rio de Janeiro, São Paulo e Salvador são as que ocupam, respectivamente, os três primeiros lugares em número de roubos e furtos no país.

Segundo a reportagem de Borges [3], em média, 150 carros são roubados ou furtados por dia no Rio de Janeiro, quatro por dia em São Paulo e 2,8 por dia em Salvador. Para a polícia, os grupos que comandam o esquema de roubo e furto articulam as ações de dentro de presídios e possuem ramificações em vários estados.

A principal utilidade dos automóveis sinistrados é a transformação em dublês (ou clones), passando por alterações no chassi, placas e outros dados para venda ilegal em outros estados, é claro que parte destes serve para a prática de crimes, como o transporte de quadrilhas durante assaltos e o transporte de drogas.

O maior problema relacionado ao crime é o risco de vida que os ocupantes do veículo são expostos, principalmente quando se há uma tentativa reação contra os criminosos, que ocorre pela incerteza se o bem perdido será recuperado pelas autoridades e em boas condições.

Diante deste cenário, é necessária uma solução que permita a geolocalização em tempo real contando com mecanismos de bloqueio que podem ser controlados remotamente, o que geraria tranquilidade aos que sofrerem roubo/furto de dos seus veículos, eliminando a ânsia de reação em um primeiro momento.

1.2 Trabalhos Relacionados

Durante o levantamento bibliográfico realizado para obtenção de trabalhos correlatos foram encontrados três projetos de relevância na categoria de alarme automotivo inteligente, alguns utilizando inclusive comunicação remota inter-dispositivos e hardware open-source.

Nascimento [12] descreve um projeto de sistema de alarme automotivo cuja principal funcionalidade é a detecção de um evento sonoro no interior do veículo e, realizando as conversões necessárias, sinaliza a um micro controlador o evento ocorrido. O micro controlador por sua vez está programado para acionar um dispositivo que realiza uma chamada telefônica destinada ao aparelho celular do proprietário do veículo de modo a alertá-lo sobre a anomalia, o objetivo geral é a detecção do choro de uma criança ou barulho de um animal esquecido dentro do veículo.

Em toda extensão do trabalho verificou-se uma abordagem bastante prática, com experimentos e construção de circuitos, além da exibição do funcionamento do protótipo, além disso, foi realizado um fundamental teórico bastante sucinto que se focou principalmente nos tipos de microfone. O ponto negativo ficou por conta da ausência de modelagem do problema, não existem diagramas UML ou similares que representem as funcionalidades, projeto de classes e sequência, que são essenciais para qualidade do projeto.

Em sua monografia, Martins [9] propõe um sistema de segurança veicular para ser integrado a um alarme já existente no veículo, este sistema objetiva além de avisar ao proprietário via SMS/GPRS quando o alarme comum for acionado, detectar quando algum objeto, criança ou animal de estimação for esquecido sobre o banco verificando a presença do peso sobre este.

O autor promove uma explanação sobre a motivação do trabalho baseado em dados jornalísticos e casos verídicos de esquecimento de crianças pelos pais e quais as consequências

cias biológicas trazidas a um indivíduo se exposto às condições ambientais semelhantes a de um carro fechado exposto ao sol.

O projeto se baseia na plataforma Arduino, utilizando a filosofia de hardware livre, modularização de componentes e desenvolvimento simplificado de protótipos para posterior produtização, o autor optou pela utilização de uma placa Arduino Mega com módulo externo.

Foram disponibilizados os passos realizados no projeto, um modelo esquemático de testes além de anexar o código fonte aplicado na placa controladora todavia há a ausência de esquemas de modelagem, principalmente baseados em UML, além da falta do esquema eletrônico geral, o que facilitaria bastante a análise das limitações e aplicação de possíveis melhorias em projetos futuros.

O trabalho de Marquez [7] se refere à criação de um sistema de autenticação, onde o usuário recebe uma mensagem desafio em um módulo portátil e a responde, para assim poder utilizar o veículo (funcionamento do motor e partes elétricas). Uma das funções deste dispositivo é o desligamento do veículo caso a central não detecte que o módulo portátil nas proximidades do veículo, se assemelhando às técnicas utilizadas nos alarmes veiculares mais modernos.

Por modularizar os componentes e estes se comunicarem utilizando radiofrequênciа, se fez necessária a utilização de criptografia para que as mensagens trocadas não fossem interceptadas, interpretadas e posteriormente injetadas, em uma tentativa de fraudar o sistema.

Apesar de descrever textualmente de modo detalhado cada módulo desenvolvido, no decorrer o trabalho não traz modelagem de requisitos e de projeto baseada em UML, apenas diagramas resultantes da fase de implementação, além disso, uma interface mais amigável com o usuário seria muito bem vinda, neste ponto, o desenvolvimento de trabalhos futuros substituindo o módulo portátil por um smartphone seria promissor.

1.3 Contribuições

Este trabalho apresenta o desenvolvimento de um sistema de segurança veicular com uso de GPS, cujas principais funcionalidades são: a obtenção da posição atual de um

automóvel e a possibilidade de seu desligamento remoto por meio de uma interface Web.

O processo de elaboração do sistema utilizou a metodologia descritas em Wolf [14] voltada a sistemas embarcados onde se aplicam alguns conceitos de Engenharia de Software para concepção de plataformas de tempo real. Com isto foram gerados artefatos de modelagem: diagramas UML englobando requisitos, arquitetura e componentes de hardware/software descritos de forma a sintetizar o funcionamento esperado do conjunto e que servem de base para novas implementações.

A escolha e montagem dos componentes de hardware, o software embarcado, protocolos de comunicação escolhidos e frameworks usados podem servir de guia para projetos com funcionalidades que se encaixem na mesma categoria.

1.4 Metodologia

Foi conduzida uma revisão bibliográfica relativa a projeto de sistemas embarcados a fim de aplicar padrões de projetos já consagrados e recomendados para a categoria desta implementação, após esta fase de embasamento teórico, elaborou-se a modelagem de requisitos utilizando modelo de casos de uso derivando naturalmente para a construção do diagrama de classes, diagramas de sequências e diagrama de atividades.

A elaboração de um diagrama de componentes foi feita para se obter uma melhor visão do sistema, distingindo módulo embarcado, módulo Web e interface com o usuário. Após isto, foram definidos os componentes de hardware para construção do módulo embarcado e feito estudo da documentação destes componentes que fora fornecida pelo fabricante.

Seguido a construção do módulo embarcado, realizou-se a implementação do serviço Web em conjunto com a interface com o usuário (UI), onde foi criada a estrutura que de comunicação entre módulos, realizando fluxos de controle, comando, armazenamento transitório de dados e integração com o serviço de mapas.

Finalizada a implementação dos módulos, a integração de todas as partes do sistema foi realizada juntamente com testes exploratórios, verificando o comportamento real da aplicação. O módulo embarcado foi instalado em um automóvel real e sua posição mostrada no serviço de mapas foi aferida para validar o funcionamento correto da plataforma, suas limitações de resposta, além da checagem da função de bloqueio. Os testes exploratórios

foram registrados em vídeo.

1.5 Organização

O restante texto deste trabalho está organizado da seguinte maneira:

- O capítulo 2 apresenta uma descrição teórica sobre os itens utilizados para o desenvolvimento do sistema.
- O capítulo 3 descreve os passos do desenvolvimento em si, desde os requisitos, o conceito da solução, a modelagem dos componentes e finalmente a implementação destes.
- O capítulo 4 traz os resultados obtidos com a implementação, dados de testes realizados e conclusões sobre o processo de desenvolvimento.

Capítulo 2

Fundamentação Teórica

2.1 Sistema Embarcado

Segundo Wolf [14] um sistema de computação embarcado é qualquer dispositivo que inclui um computador programável, o qual não é direcionado para resolver problemas de propósito geral, mas sim uma situação ou necessidade específica, operando geralmente com recursos limitados e executando algoritmos para resposta em tempo real.

A diferenciação entre um sistema embarcado de um sistema de propósito geral se dá por algumas características:

- *Tempo real*: a resposta do sistema deve ser dada em um tempo limite. Caso a resposta venha fora do limite de tempo aceitável, o sistema pode ser tornar instável, parar de funcionar ou realizar ações fora do previsto gerando danos catastróficos.
- *Diferentes taxas de entrada de dados*: um sistema embarcado pode receber diversas entradas de dados com velocidades diferentes, ele deve tratar esta situação e oferecer uma saída sincronizada e sem perdas.
- *Custo de produção*: todos os componentes de hardware devem ser escolhidos para minimizar o custo de produção, por isso, um projeto cuidadosamente elaborado se faz necessário.

- *Consumo de energia:* conjunto hardware/software dever ser projetado de maneira tal que o consumo de recursos energéticos seja sempre o mínimo possível. Um elevado consumo influí tanto no custo de produção quanto no custo de operação.

Um exemplo de sistema embarcado a se considerar é um roteador wireless conforme a figura 2.1, pois possui as seguintes características:

- É um computador de propósito específico: realiza funções de roteamento, controle de conexões, controle do sinal de rádio e implementa serviços da camada física a camada de rede (modelo OSI);
- Opera em tempo real: todos os pacotes que passam pelo roteador deve ser processados em tempo hábil, caso contrário a comunicação entre dispositivos que estão conectados a ele se torna impraticável;
- Custo de produção resuzido: São utilizados processadores de custo reduzido, quantidade de memória limitada e outros componentes de menor custo a fim de reduzir o preço total e facilitar a produção em massa.
- Baixo consumo de energia: O conjunto hardware/software opera com baixíssimo consumo de energia a fim de não impactar na conta de energia do consumidor, afinal é um produto que funciona, em geral, 24 horas por dia.



Figura 2.1: Um roteador wireless é um sistema embarcado

2.2 Projeto de Sistemas Embarcados

Modelos para projeto de sistemas embarcados têm sido propostos ao longo dos anos, como é caso do Embedded UML. De acordo com Martin, Lavagno e Louis-Guerin [8], autor do modelo, este é um profile UML que representa a síntese de várias ideias discutidas pela comunidade que adota UML no projeto de aplicações de tempo real.

Foram selecionadas as melhores práticas de análise e especificação de requisitos para sistemas os quais necessitam que hardware e software sejam co-projetados, introduzido um conceito de mapeamento da plataforma para gerar uma implementação otimizada de hardware e software, uma vez que conforme Wolf [14], o custo de produção e o consumo de recursos energéticos deve ser o mínimo possível.

Esta metodologia fornece uma abordagem que permite o desenvolvimento de ferramentas de análise automatizada, simulação, síntese e geração de código quanto se define um padrão UML para embarcados.

O próprio Martin, Lavagno e Louis-Guerin [8], propõe em um segundo artigo, após utilizar UML para modelagem de sistemas embarcados, novas extensões UML que cobrem lacunas não atendidas pela linguagem, como suporte a modelagem de hardware baseada em VHDL (que é uma abordagem dependente de plataforma).

É claro que na UML existem algumas limitações de análise de cenários com requisitos não funcionais, que no caso de embarcados têm prioridade igual a dos requisitos funcionais e de suma importância para a confiabilidade do sistema. Esta limitação é descrita por Espinoza, Servat e Gérard [4] que propõem práticas complementares para analisar estes requisitos não funcionais, a fim de tornar os sistemas mais eficientes de confiáveis.

Wolf [14] propõe uma abstração em alto nível dos principais passos no projeto de sistemas embarcados, são estes do topo à base:

- *Requisitos*: onde são obtidas informações sobre o que se desenvolver no sistema, obter os requisitos esperados pelo cliente/usuário e extrair apenas o que for importante para o projeto levando requisitos funcionais e não funcionais são levados em conta.
- *Especificação*: etapa onde se realiza a mapeamento dos requisitos funcionais de um sistema embarcado, quais funções ele deve fornecer ao usuário. Os Casos de Uso da UML são um meio de documentar os requisitos.

- *Projeto Arquitetural:* nesta etapa é feita a quebra do sistema em componentes de alto nível e como eles se integram e se comunicam. A UML oferece o Diagrama de Componentes para esta finalidade.
- *Projeto de componentes de hardware e software:* para cada componente definido, são mapeados seus atributos e métodos, levando a uma abordagem utilizando Orientação a Objetos. O Diagrama de Classes da UML permite fazer este tipo de representação.
- *Integração do sistema:* após a construção de cada componente de hardware e implementação dos componentes de software, estes são integrados seguindo o projeto.

Marwedel [10] inclui etapas de teste dentro das etapas de projeto, designando o fluxo como modelo-V. As etapas de teste são:

- *Testes unitários:* Teste individual para cada unidade mínima funcional do sistema, com isto é possível verificar o comportamento de cada parte e, em caso de defeitos, corrigi-los antes de qualquer integração, poupando tempo de desenvolvimento uma vez que seria dispendioso, depois do sistema pronto, rastrear e econtrar o problema.
- *Testes de integração:* Verificação da comunicação entre os diferentes componentes quanto interligados. Esta etapa é válida pois mesmo que os componentes funcionem individualmente e separados, sempre que se integram partes um novo comportamento é gerado, o que pode levar a erros inesperados.
- *Aceitação e Uso:* Verifica se o comportamento do sistema está dentro do esperado nas especificações, é um teste do sistema completo e em funcionamento, parar confirmação que todos os requisitos elicitados estão sendo atendidos.

2.3 Microcontroladores

Um microcontrolador é uma unidade de processamento que, diferente de um microprocessador, traz todos os módulos necessários ao seu funcionamento (como memória volátil, memória somente leitura, blocos de entrada e saída, conversores analógico-digital e digital-analógico, linhas para troca de dados com componentes externos) no interior de um único chip, podendo assim ser programado para executar uma rotina de propósito específico.

Realizando uma comparativa com os microprocessadores, pode-se constatar que os microcontroladores operam em uma frequência muito baixa no entanto adequada a maioria das aplicações as quais são designados, como por exemplo: controlar uma esteira, uma caldeira ou uma máquina. Devido a esta característica, o consumo de energia é pequeno, normalmente na casa dos miliwatts, além disso os microncontroladores podem entrar em modo de espera, aguardando por um evento externo como o pressionar de uma tecla ou acionamento de um sensor.

Uma outra característica antagônica em relação aos microprocessadores é que, enquanto neste últimos é feito um superdimensionamento de recursos sendo limitado pela faixa financeira que o usuário pode investir, em um projeto com microcontroladores o superdimensionamento é um erro de projeto, um desperdício de recurso que reflete diretamente no preço do equipamento final, sendo multiplicado no caso de uma produção em larga escala.

As aplicações deste componente compreendem principalmente automação e controle de produto periféricos, como sistemas de controle de motores automotivos, máquinas industriais, de escritório e residenciais, brinquedos, sistemas de supervisão e outros. Por reduzir o tamanho, custo e consumo de energia se comparados ao microprocessadores, se tornam uma alternativa eficiente para controlar processos e aplicações. A figura 2.2 mostra um microcontrolador da fabricante Atmel, pode-se observar que ele possui um único encapsulamento com todos os periféricos necessários ao seu funcionamento embutidos.

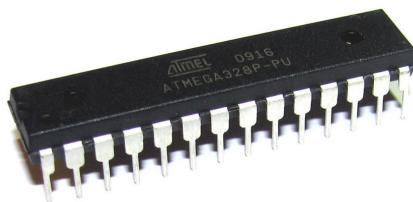


Figura 2.2: Microcontrolador Atmel

2.4 Plataforma Arduino

Arduino é uma plataforma para prototipagem eletrônica que utiliza o conceito de hardware livre. Neste conceito deve-se prover um lançamento irrestrito de informações sobre o

projeto de hardware como diagramas eletrônicos, estrutura de produtos, layout de placas de circuito impresso, rotinas de baixo nível e qualquer outra informação necessária para um construção from-scratch do projeto.

A linhagem mais utilizada da placa microcontroladora Arduino (modelos Duemilanove e Uno) foi concebida com base em um microncontrolador Atmel AVR montado em placa única, possui suporte de entra/saída embutido, uma linguagem de programação padrão baseada em C/C++.

A plataforma surgiu para que amadores, entusiastas e outras pessoas que não teriam acesso ao controladores e ferramentas mais sofisticadas, pudessem criar seus projetos, tornando-os acessíveis, flexíveis e com baixo custo. Uma placa Arduino, em geral, possui os seguintes elementos em sua construção:

- Um microcontrolador
- Placa base com reguladores de voltagem adequados ao microcontrolador.
- Linhas de Entrada/Saída digitais e analógicas.
- Linhas de comunicação serial.
- Interface USB para programação e interação com um computador hospedeiro.

Seguindo estas características comuns, ao decorrer da existência do projeto foram desenvolvidas diversas placas microncontroladoras, cada uma carrega um codinome e especificações de hardware próprias. A tabela 2.1 descreve algumas placas da família Arduino.

Modelo	Clock	E/S Digital	E/S Analógico	Alimentação	Flash
Arduino Due	84MHz	54	12	7-12V	512 KB
Arduino Leonardo	16MHz	20	12	7-12V	32 KB
Arduino Uno	16MHz	14	6	7-12V	32 KB
Arduino Duemilanove	16MHz	14	6	7-12V	32 KB
Arduino Pro	8MHz	14	6	3.3-12V	32 KB
Arduino Mega	16MHz	54	16	7-12V	256 KB
Arduino Mini 05	16MHz	14	6	7-9V	32 KB
Arduino Fio	8MHz	14	8	3.3-12V	32 KB
LilyPad Arduino	8MHz	14	6	2.7-5.5V	32 KB

Tabela 2.1: Modelos de placa Arduino

Uma possibilidade bastante interessante para projetos que utilizam a plataforma Arduino é a expansão da placa microcontroladora através da agregação de novos hardwares, estes chamados de shields (do inglês concha).

2.4.1 Arduino Shields

Placas Arduinos e outras baseadas no projeto utilizam expansões de hardware chamadas shields. São placas de circuito impresso fixadas ao topo da placa microcontroladora através dos pinos de conexão e se comunicam com a unidade principal através dos canais de Entrada/Saída analógicos ou digitais, ou ainda, através do canal de comunicação serial.

Caso não utilizem a mesma pinagem, pode-se empilhar diversos shields na mesma placa controladora. A idéia destas expansões é criar componentes com determinada especialização, muito similar ao conceito de framework em software. Existem shields para as mais diversas aplicações como:

- Conectividade em rede Ethernet
- Conectividade em rede Zigbee
- Conectividade em rede Wireless 802.11
- Controle de motores
- Conexão com rede celular GPRS
- Funcionalidade GPS
- Middleware Android
- Controle de Relés
- Bluetooth
- Sintetizador de Voz

A figura 2.3 mostra uma placa Arduino Uno conectada a dois shields para controle de motores. Pode-se observar a facilidade de conexão entre os componentes, uma vez que utilizam um layout de pinagem padronizado.

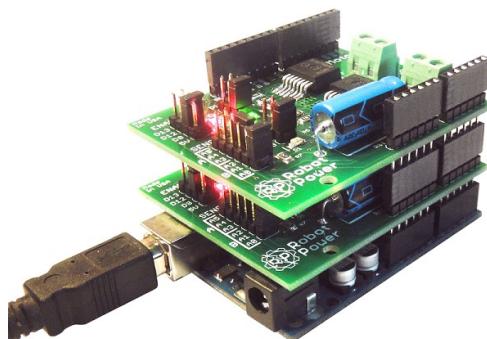


Figura 2.3: Placa Arduino com shields conectados

2.5 Placa Arduino Uno

Para realizar a implementação deste trabalho, optou-se por utilizar uma placa Arduino Uno, pois atendia todos os requisitos de hardware necessários sem haver superdimensionamento, seguindo a regra para projeto de sistemas embarcados, além disso seu layout permite a conexão da maioria dos shields disponíveis no mercado sem a necessidade de adaptação.

A placa Arduino Uno é baseada no microcontrolador ATmega328, possuindo 14 pinos de entrada/saída digitais (dos quais 6 podem ser utilizados como saída PWM), 6 entradas analógicas, cristal de cerâmica com 16 MHz de clock que define a frequência de operação do microcontrolador, conexão USB, pino de alimentação externa padrão barrel, botão de reset e conexão ICSP. A placa possui todos os circuitos auxiliares para o funcionamento do microcontrolador, bastando ao usuário conectá-la a um computador via cabo USB para começar a utilizá-la, além disso o processo de gravação de rotinas é facilitado por haver um bootloader¹ que captura automatiza este passo (Arduino [2]).

A figura 2.4 exibe uma tomada frontal da placa Arduino Uno onde é visível o microcontrolador ATmega328 em encapsulamento SMD. É observável que a placa possui um segundo controlador, um ATmega16U2 utilizado como conversor USB-serial o que permite sua programação via porta USB, além dos circuitos de controle de clock, alimentação, reset e conexão aos pinos de E/S. Os pinos da placa são descritos com textos impressos na superfície.

¹Bootloader é o gerenciador de inicialização do microcontrolador.

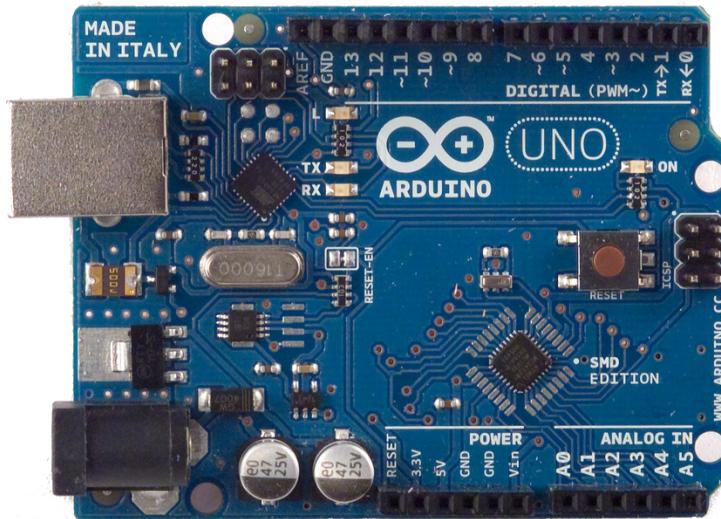


Figura 2.4: Placa Arduino Uno em detalhes

2.5.1 Especificações Arduino Uno

A tabela 2.2 mostra um resumo das especificações operacionais da placa Arduino Uno.

Microcontrolador	ATmega328
Voltagem de Operação (recomendada)	7-12V
Voltagem de Operação (limites)	6-20V
Pinos de E/S Digitais	14 (onde 6 permitem saída PWM)
Pinos de entrada analógicos	6
Corrente DC por pino de E/S	40 mA
Corrente DC para o pino 3.3V	50 mA
Armazenamento em Flash	32 KB, onde 0.5 KB são usados pelo bootloader
Memória RAM (SRAM)	2 KB
Memória EEPROM	1 KB
Frequência de clock	16 MHz

Tabela 2.2: Resumo das Especificações - Arduino Uno

2.5.2 Alimentação da Placa Arduino Uno

Um Arduino Uno pode ser alimentado pela conexão USB ou uma fonte externa e a placa faz a seleção automática de fonte de entrada. Uma alimentação com tensão inferior a 7V pode fazer com que o pino de saída de 5V para o microcontrolador forneça uma tensão inferior a descrita causando instabilidade no funcionamento. Por outro lado, se colocada uma

tensão de entrada maior que 12V, o regulador de voltagem pode sofrer superaquecimento, danificando a placa e diminuindo sua vida útil.

No desenvolvimento deste trabalho foi utilizada, para testes em bancada, uma fonte regulada com saída de 12V e 1850mA de corrente máxima, retratada na figura 2.5.



Figura 2.5: Fonte de alimentação para testes em bancada

Para a aplicação em automóveis não se faz necessário o uso de regulação de voltagem já que os veículos utilizam uma voltagem padronizada de 12V, com variações até o máximo de 14V. Apenas é preciso encontrar um ponto de alimentação permanente dentre os cabos do painel e utilizar um cabo com terminal adequado para ligação com a placa Arduino Uno.

Após passar pelos circuitos de alimentação da placa, existem alguns pinos que fornecem alimentação tanto para periféricos quanto para os shields, são estes:

- **Pino 5V:** saída de tensão regulada e estabilizada em 5 volts para alimentar o microcontrolador e outros periféricos como shield, servo-motores de pequeno porte e relés.
- **Pino 3V3:** saída de tensão regulada e estabilizada em 3.3 volts e suporta uma carga máxima de 50 mA. Utilizada principalmente pelo conversor USB-serial e shields que

trabalham com comunicação serial TTL.

- **GND:** pino terra, referencial 0 volts para os circuitos.
- **VIN:** mesmo ponto da entrada de alimentação externa, refletindo o mesmo valor da tensão de alimentação do conector jack (barrel).

2.5.3 Entrada e Saída

O Arduino Uno possui 14 pinos digitais que podem ser usados como entrada ou saída, operando a 5 volts e suportando 40mA de corrente máxima. Existem funções na biblioteca de programação padrão para facilitar a utilização destes pinos, são elas:

- `pinMode()`: Configura o modo de operação do pino para entrada ou saída.
- `digitalWrite()`: Em modo saída, manipula o estado do pino para HIGH (5V) ou LOW (0V).
- `digitalRead()`: Em modo entrada, lê o estado do pino se está em HIGH ou LOW.

Dentre estes pinos, alguns possuem funções especiais como:

- Serial: 0 (RX) e 1 (TX). Usados para receber (RX) e transmitir (TX) dados usando padrão serial TTL. São interconectados ao controlador USB.
- Interrupção externa: 2 e 3. Podem ser configurados para acionar uma função de interrupção em LOW, borda de subida, borda de descida ou mudança no valor, configurados na função `attachInterrupt()`.
- PWM: 3,5,6,9,10 e 11. Permite saída PWM² de 8 bits, simulando uma saída analógica através da função `analogWrite()`.
- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Pinos para comunicação SPI³.

²Variação de Largura de Pulso: Técnica onde se utiliza um sinal digital para variar o valor da transferência de potência a uma carga de alimentação analógica, simulando, por exemplo uma variação na tensão de alimentação.

³Interface de Periféricos Síncrona, padrão para comunicação entre periféricos no modo mestre-escravo

- LED: 13. Este pino possui um diodo emissor de luz acoplador permitindo testes de entrada/saída com facilidade.

2.5.4 Programação

A programação do microcontrolador ATmega328 do Arduino Uno pode ser feita através da IDE Arduino, mostrada na figura 2.6. A linguagem de desenvolvimento é C/C++ com recursos exclusivos desenvolvidos com base na linguagem Wiring. O microcontrolador vem com um bootloader pré-gravado, o que permite o upload de novos códigos com o pressionar de um botão, sem uso de hardware para programação externa, como no caso de gravadores de flash.

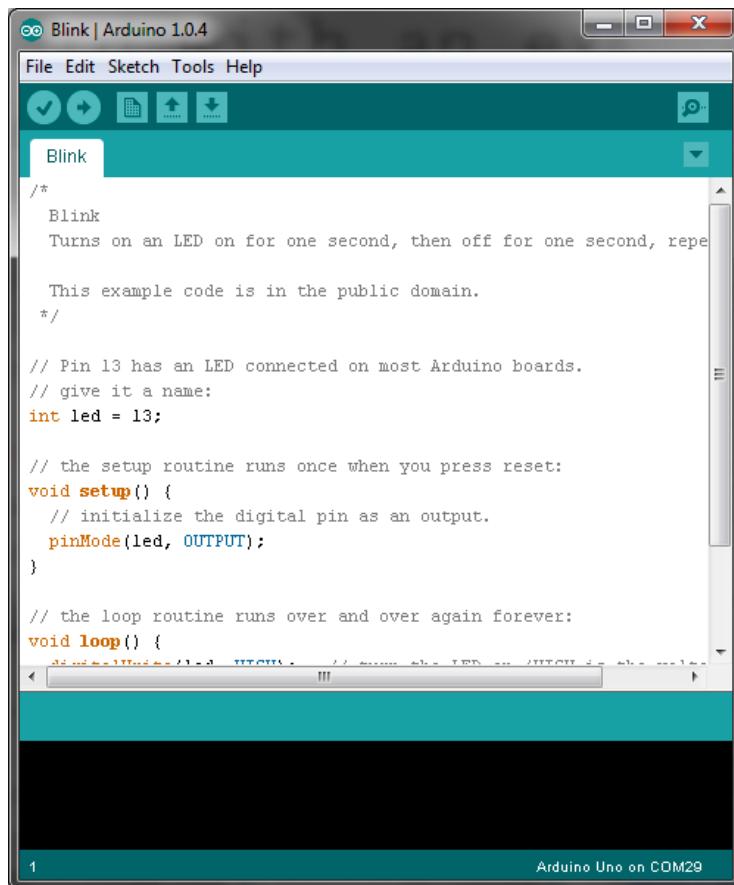


Figura 2.6: IDE Arduino

É possível realizar a programação do microcontrolador por meio da interface ICSP (In-Circuit Serial Programming), entretanto esse estratégia não foi utilizada para este trabalho,

mais informações consultar ICSP [6].

Sempre que é feita uma gravação de rotina no controlador é feita uma reinicialização do circuito, forçando a sobreescrita de dados na memória RAM. A reinicialização também pode ser feita através do botão reset ou mesmo de um circuito interligado ao pino RESET controlado via software.

2.6 Tecnologia GPS

O termo GPS foi extraído da designação NAVigation System with Time and Ranging Global Positioning System - NAVSTAR GPS, sistema inicialmente voltado às operações militares, auxiliando na navegação e é em síntese um sistema de rádio navegação que fornece as coordenadas bi ou tridimensionais de pontos no terreno, além da velocidade e direção do deslocamento entre pontos.

De acordo com Albuquerque e Santos [1], para o funcionamento do GPS, existe uma constelação composta por 24 satélites na órbita terrestre, distribuídos em 6 planos orbitais igualmente espaçados, com 4 satélites em cada plano a uma altitude de aproximadamente 20200 km. Os planos orbitais estão inclinados a 55° em relação ao Equador e o período orbital é de aproximadamente 12 horas siderais. Com esta configuração fica garantido que no mínimo 4 satélites visíveis na superfície da Terra em qualquer horário do dia. A figura 2.7 mostra uma idéia conceitual desta configuração.

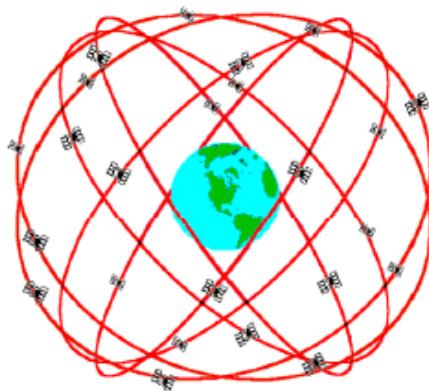


Figura 2.7: Modelo conceitual da constelação de satélites GPS

Além das atividades de navegação, o sistema GPS também auxiliar na realização de

levantamentos geodésicos e topográficos, operando de forma ininterrupta, independente das condições meteorológicas mesmo que estas possam afetar na precisão dos dados.

2.6.1 Modelo de Posicionamento GPS

Para identificar a posição de pontos de interesse, o sistema GPS utiliza as coordenadas de seus satélites, que são referenciadas a um sistema geodésico, o mesmo utilizado no receptor GPS para processar os dados recebidos e determinar as coordenadas do ponto referido. A figura 2.8 mostra este processo, onde o ponto é calculado através das referências de 5 satélites, o ponto P é obtido por triangularização das coordenadas.

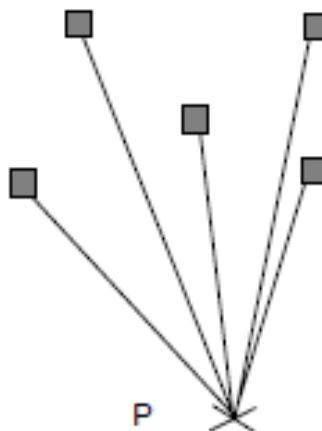


Figura 2.8: Conceito para cálculo da coordenada de um ponto via GPS

Um sistema GPS é composto por três elementos principais conforme a figura 2.9, são eles:

- Estação de Controle: são distribuídas em torno da Terra, próximas da linha do Equador. Dentre suas funções, realizam o monitoramento e controle dos satélites, determinam o tempo GPS, prevêem as efemérides, calculam as correções dos relógios e atualizam as mensagens de navegação dos satélites.
- Satélite GPS: propaga as mensagens GPS com suas respectivas coordenadas para que os receptores possam calcular sua posição.
- Receptor: usuário do sistema, equipamento que se utiliza dos satélites para obter a posição de determinado ponto.



Figura 2.9: Elementos do sistema GPS

2.7 Arduino GPS Shield Kit

Para adicionar o suporte a rastreamento via GPS em tempo real a este trabalho, um hardware que implemente tal funcionalidade se faz necessário. Neste âmbito, o conceito de shield da plataforma Arduino oferece um gama de componentes que atendem esta necessidade.

Após realizar uma pesquisa de mercado, optou-se pelo uso do GPS Shield Retail Kit da fabricante Sparkfun (Shield [13]), que se mostrou financeiramente mais viável, além disso o fabricante disponibiliza um rico arcabouço documental, programas de exemplo e esquemáticos de hardware. A figura 2.10 mostra os elementos do kit GPS.

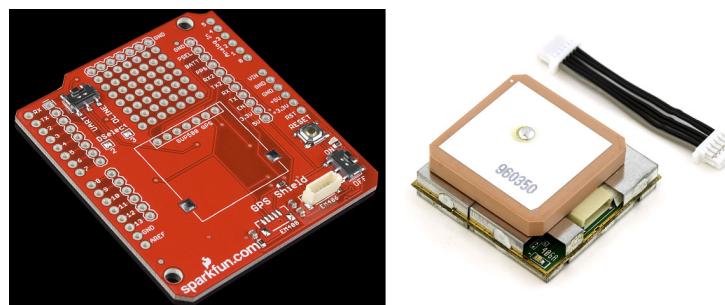


Figura 2.10: Kit GPS

O kit é composto de dois elementos principais, são eles:

- Shield (ou placa base): É uma placa de circuito impresso com layout padrão Arduino Uno contendo reguladores de tensão para 3.3 volts, leds indicadores, chave liga/desliga, botão reset e um conector padronizado para diversos modelos de módulos GPS. Esta placa é a peça chave para integração entre o Arduino Uno e o módulo GPS, pois isenta o projetista de realizar qualquer solda ou conexão adicional (salvo casos em que se faz necessário mudar os pinos de comunicação serial), permitindo que este adicione ou remova a funcionalidade GPS ao seu projeto com extrema facilidade.

A figura 2.11 mostra o esquemático do GPS Shield.

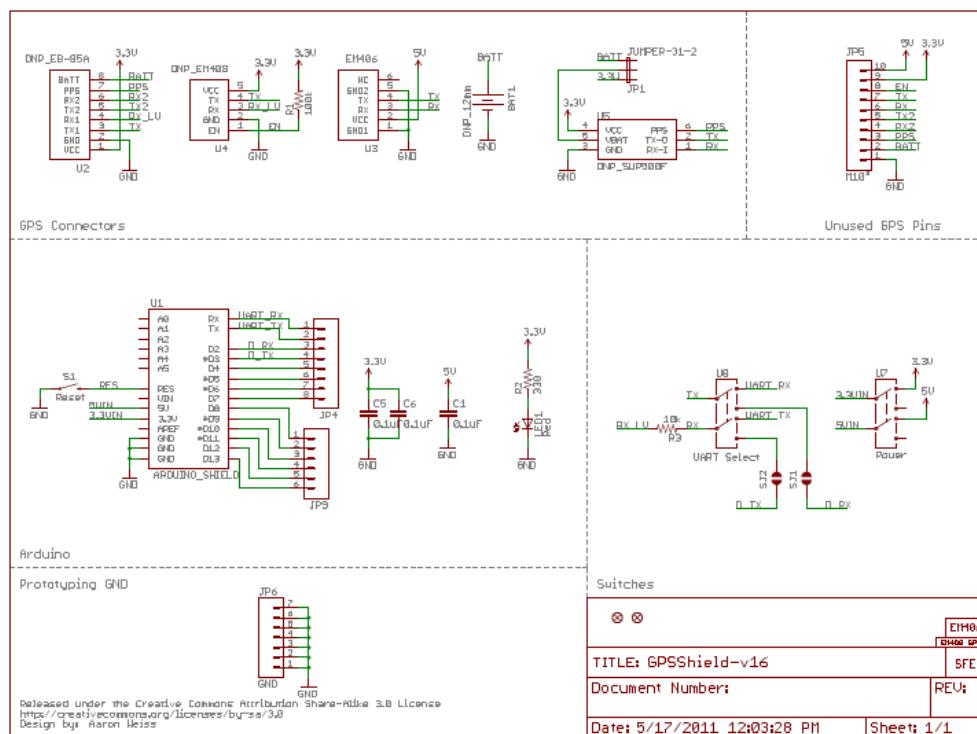


Figura 2.11: Esquema elétrico do GPS Shield

- Módulo receptor GPS USGlobalSat EM-406A: Realiza a função de captura das mensagens GPS e cálculo de posicionamento global. Possui todos os componentes integrados, inclusive antena, regulador de voltagem e LED de status. É conectado ao GPS Shield por meio de um cabo de 4 vias. A tabela 2.3 mostra os dados operacionais do módulo EM-406A.

Canais de recepção	20
Sensibilidade	-159 dBm
Precisão	10 metros, 5 metros com WAAS
Hot Start	1 segundo
Warm Start	38 segundos
Cold Start	42 segundos
Consumo	70 mA em 4.5-6.5 volts
Protocolo de Saída	NMEA 0183 e SiRF
Dimensões	30mm x 30mm x 10.5mm
Peso	16 gramas

Tabela 2.3: Especificações operacionais do módulo GPS EM-406A

O módulo EM-406A trafega o formato de dados configurado de fábrica em sua resposta ao microcontrolador, este formato é definido no protocolo NMEA 0183 ACII versão 3.01. A mensagem segue a estrutura mostrada na figura 2.12 e cada seção dela é descrito na tabela 2.4.

\$GPRMC,092204.999,A,4250.5589,S,14718.5084,E,0.00,89.68,211200,,A*25<CR><LF>
1 2 3 4 5 6 7 8

Figura 2.12: Mensagem NMEA 0183

Campo	Nome	Exemplo	Descrição
1	Tempo UTC	060932.448	Horário UTC no formato hhmmss.sss
2	Status	A	'V' = GPS Aquecendo; 'A' = Dados Válidos
3	Latitude	2447.0959	Latitude no formato ddmm.mmmm
4	Indicador N/S	N	Hemisfério, 'N' = Norte, 'S' = Sul
5	Longitude	12100.5204	Longitude no formato dddmm.mmmm
6	Indicador E/W	E	Hemisfério, 'E' = Leste, 'W' = Oeste
7	Velocidade	000.0	Velocidade em nós (000.0 999.9)
8	Data UTC	211200	Data UTC de uma posição fixa no formato, ddmmyy

Tabela 2.4: Descrições das seções da mensagem NMEA 0183

Capítulo 3

Desenvolvimento do Protótipo

3.1 Modelo Conceitual

Nesta fase é necessário fechar o escopo do projeto, capturar os requisitos e principais comportamentos esperados. No cenário de operação, cada automóvel a ser verificado possui um módulo embarcado instalado e responsável por enviar sua localização geográfica atual obtida de um GPS. Estes dados são enviados a um servidor web por meio de rede GPRS de uma operadora de telefonia móvel, o servidor web captura as informações e organiza em uma base de dados.

Na outra ponta do sistema está o usuário, portanto deve ser oferecida uma interface para que ele possa localizar seus veículos e opcionalmente realizar o desligamento remoto dos mesmos. Todo acesso de usuário será feito via navegador Web. A Figura 1 mostra o modelo conceitual do sistema.

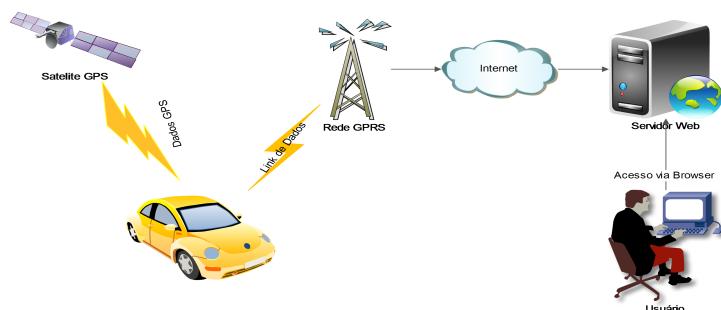


Figura 3.1: Visão de operação do sistema e os elementos envolvidos

3.2 Especificação de Requisitos

Foram identificados dois atores do sistema (agentes externos): um “usuário comum” e um “administrador”. Os requisitos são especificados sob a forma de casos de uso UML. A figura 2 retrata os casos de uso de cada ator identificado.

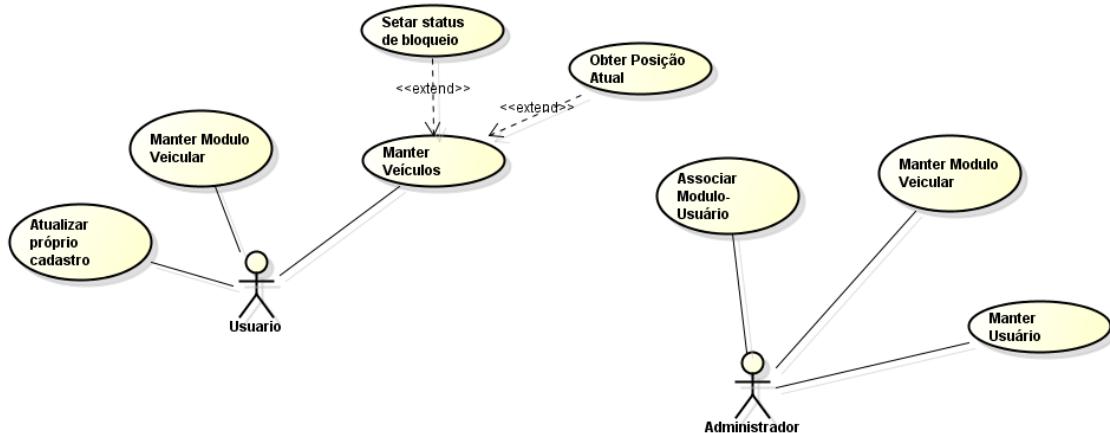


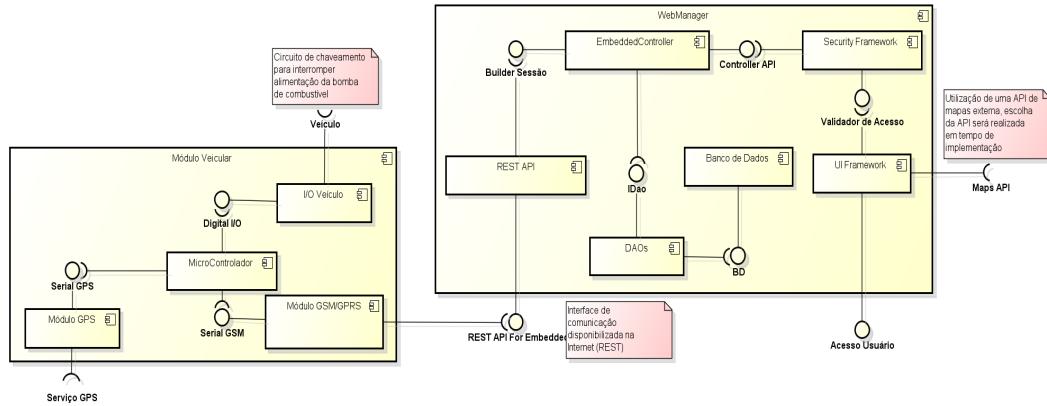
Figura 3.2: Casos de uso do sistema

3.3 Arquitetura do Sistema

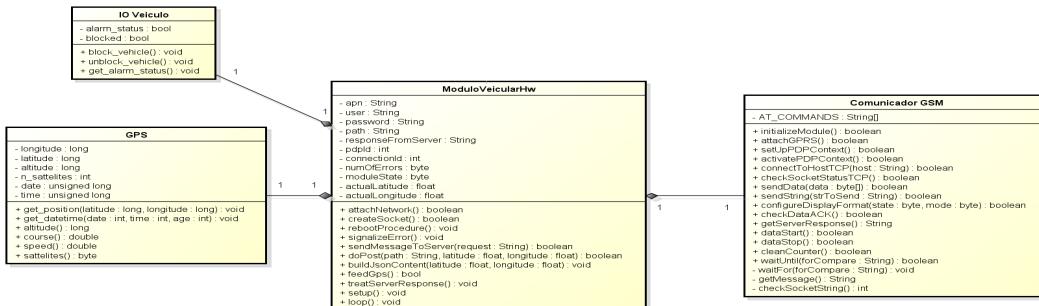
Os blocos iniciais do projeto juntamente com os casos de uso e suas descrições nos permitem fazer uma quebra do sistema em componentes independentes para arquitetura de sistemas de tempo real de acordo com Mendes [11]. A Figura 3 mostra o diagrama de componentes da UML gerado nesta etapa.

3.4 Módulo Veicular Embarcado - Modelagem

A principal vantagem em modelar os componentes de hardware com o digrama de classes é observada no momento de escrever funções ou bibliotecas para comunicação entre componentes, pois como o mapeamento de métodos necessários já está feito, o desenvolvimento é focado apenas nestes, evitando a criação de código não utilizado. No caso deste

**Figura 3.3:** Arquitetura do Sistema

projeto, o componente GSM fornece uma gama de funcionalidade que não são utilizadas, como funções de chamada de voz, mensagem SMS e outras, portanto não foi despendido tempo com sua implementação. A Figura 4 mostra o projeto do módulo veicular embarcado.

**Figura 3.4:** Classes do Módulo Veicular

3.5 Módulo Veicular Embarcado - Hardware

Após a análise do diagrama de classes, foram identificados os componentes de hardware, são estes:

- Módulo GPS.

- Módulo GSM.
- Módulo de I/O.

Para implementação em hardware foi escolhida a plataforma Arduino, pois traz a filosofia do hardware open-source com seus diagramas disponíveis livremente na Internet e uma comunidade ativa no desenvolvimento de soluções.

Como placa microcontroladora, o Arduino Uno foi selecionado, pois oferece poder de processamento suficiente para o projeto, preço acessível e dimensões físicas propícias para integração veicular. A placa traz um microcontrolador Atmega 328 da Atmel, que possui 32 kbytes de memória flash, 2 kbytes de memória RAM, pinos de I/O digital e analógicos, comunicação serial e opera a um clock de 16 MHz. A figura 5 mostra um exemplar da placa Arduino Uno.



Figura 3.5: Placa Arduino Uno

A plataforma Arduino oferece diversos módulos de hardware plugáveis para as mais diversas funções, que são chamados de Shields. Estes módulos objetivam ser plug-and-play, ou seja, procuram poupar o máximo o trabalho de soldagem e aplicação de componentes discretos.

Para função de hardware GPS, o Shield GPS Sparkfun foi escolhido, este shield traz um módulo GPS EM 406 com precisão de 5 metros na medida de posição, comunicável por interface serial RS 232 ou TTL. A figura 6 mostra o shield GPS juntamente com o módulo GPS EM 406.

Para comunicação com o servidor, se fez necessária uma alternativa que utilizasse envio de dados sobre a rede de celular, pois esta rede possui altíssima cobertura em área urbana, foco do projeto. A troca de dados se dá sobre o protocolo GPRS, que especifica um máximo de 80 kbps para download e 20 kpbs para upload de dados.

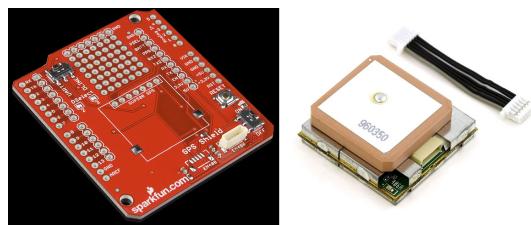


Figura 3.6: GPS Shield e Módulo EM 406

Uma das soluções em shield para Arduino é o Cellular Shield Sparkfun, que traz um módulo SendTrue SM5100B como interface com a rede. As especificações do módulo incluem:

- Quad-Band 850/900/1800/1900 MHz
- Controle via comandos AT
- Alimentação: 3,2 a 4,2V
- Baixo consumo: 350,0mA (*consumo médio em transmissão*)
- Temperatura de operação: -10 a 85 graus Celsius.

A figura 7 mostra o Cellular Shield Sparkfun com módulo SM5100B que foi aplicado no projeto. O shield foi utilizado com antena quad-band que não está contemplada na figura.



Figura 3.7: Cellular Shield com SM5100B

O módulo SM5100B não possui uma biblioteca do fabricante com funções pré-definidas para sua operação, mas é operável por meio de comandos AT especificados em suas referências. Entretanto, o uso direto de comandos AT insere duplicações no código fonte, eleva o consumo de memória, torna bastante complexo o controle de estados e a recuperação de

erros. Para contornar esta questão, foi implementada uma classe de comunicação com o módulo, que é uma subclasse de SoftwareSerial, que por sua vez é uma classe de comunicação serial RS 232 com controle via software. A figura 8 mostra o diagrama da classe de comunicação e a herança de SoftwareSerial.

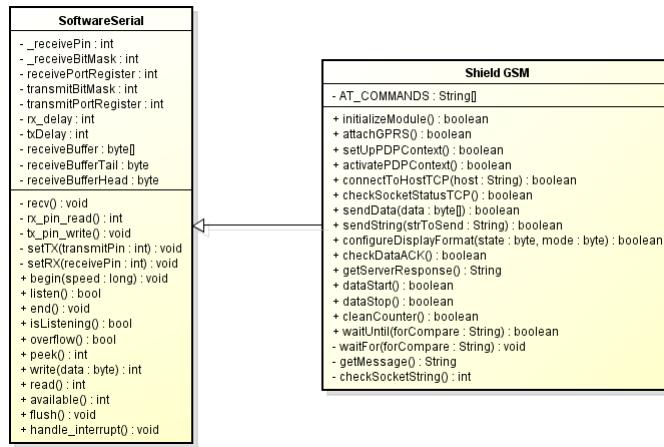


Figura 3.8: Classe de comunicação com SM5100B

Um dos requisitos funcionais do projeto prevê o bloqueio remoto de um veículo por meio do desligamento do motor. Após um estudo dos circuitos elétricos veiculares e pesquisa sobre métodos de bloqueio mais utilizados, chegou-se à conclusão que a melhor alternativa é a interrupção da alimentação elétrica da bomba de combustível, esta estratégia é utilizada pelos módulos de alarme mais confiáveis do mercado para implementar o desligamento do veículo.

Como a bomba de combustível opera em um circuito que gera corrente entre 5 e 10 ampéres, foi preciso utilizar um componente que trabalhe com esta faixa de carga. Um relé foi a solução aplicada, mais especificamente o circuito normalmente fechado do relé, cuja carga suportada é de 10 A. Para acionamento do relé, um circuito auxiliar que utiliza o sinal de uma das saídas digitais do Arduino Uno foi usado, assim, o sinal de baixa corrente da porta digital controla o acionamento da bobina de um relé, que aciona mecanicamente o circuito normalmente fechado ligado a bomba de combustível, interrompendo a alimentação da mesma quando solicitado. A figura 9 mostra o módulo de I/O com relé integrado.

Após a seleção dos componentes, o módulo veicular foi montado. Os shields foram montados com facilidade já que existe uma conexão física padrão, apenas alguns ajustes



Figura 3.9: Módulo de I/O com veículo

foram necessários: no módulo GPS, os pinos de comunicação serial (TX e RX) coincidiram com os pinos do módulo GSM (pinos 2 e 3 digital da placa Arduino), para solucionar a questão, um jumper foi feito de modo a conectar a transmissão e recepção do GPS nos pinos 4 e 5 da placa Arduino. A figura 10 mostra o módulo embarcado montado.

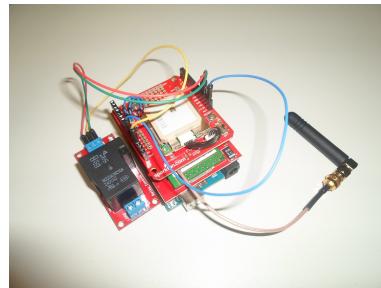


Figura 3.10: Módulo veicular montado

O módulo veicular embarcado se comunica com o servidor web através de requisições HTTP POST, trazendo em seu payload dados estruturados no padrão JSON. O adoção do método POST se deu pois o permite um maior tamanho de mensagem, e como nesta mensagem são enviados dados autenticação, estes valores não seriam gravados em texto plano nos logs, como acontece com o GET. A estrutura JSON traz os seguintes campos:

- idModule: código serial do módulo usado como identificador único na base de dados do sistema web.
- codAccess: usado como senha para validação da autenticidade do módulo.
- latitude: latitude geográfica atual obtida pelo GPS.

- longitude: longitude geográfica atual obtida pelo GPS.
- alarm: valor representando se o alarme do veículo foi acionado, 0 se falso 1 se verdadeiro. Não utilizado no momento.

A figura 11 mostra a mensagem enviada pelo módulo veicular a um servidor de testes, é possível observer o cabeçalho HTTP simulando um browser com engine Mozilla/5.0 e a sinalização do uso de dados estruturados JSON.

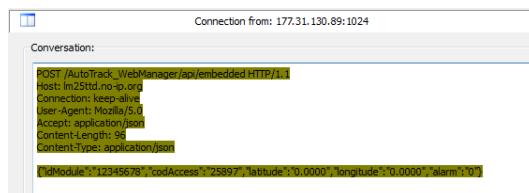


Figura 3.11: Requisição HTTP feita pelo módulo veicular

3.6 Módulo Web (WebManager) - Introdução

O módulo web, ou WebManager, é a aplicação que roda na internet responsável por receber e armezenar os dados vindos do módulo embarcado, transmitir comandos para este e permitir a interação do usuário com o sistema. O WebManager conta com um webservice RESTful que é responsável pela comunicação com o módulo embarcado.

Um webservice é uma solução adotada na integração de sistemas que funcionam em diferentes arquiteturas de hardware, normalizando a troca de informações para um formato (linguagem) universal, em geral uma combinação de uso do protocolo HTTP e dados estruturados XML/JSON.

A técnica REST permite fazer mapeamentos de métodos diretamente em URL com correspondência aos métodos HTTP, por exemplo, uma método de uma classe Java pode ser mapeado para uma URL chamada via método GET.

A implementação do WebManager como um todo foi feita utilizando a linguagem Java, por possuir um nível de maturidade muito alto para aplicações e serviços Web, contando com uma extensa gama de frameworks para os mais diversos fins, como mapeamento objeto-relacional, interface com o usuário, webservices e outros.

Como banco de dados, o PostgreSQL foi adotado pois possui um sofisticado mecanismo de bloqueio, suporta tamanhos ilimitados de linhas, bancos de dados e tabelas (até 16TB), aceita vários tipos de sub-consultas, possui mais tipos de dados e conta com um bom mecanismo de FAILSAVE (Segurança contra falhas, por exemplo no desligamento repentino do sistema).

3.7 Módulo Web (WebManager) - Modelagem

A arquitetura da modelagem do WebManager foi baseada no Pattern MVC que segundo Mendes [11] estabelece o desenvolvimento de software em camadas: classes de Entidade (Models), classes de controle (Control) e classes de interface com o usuário (View).

O modelo MVC é consagrado, pois permite o desacoplamento das camadas de software facilitando a manutenção e diminuindo a repetição de código, as vantagens de sua utilização ficam bastante claras quando existe a necessidade da alteração de código, por exemplo, caso haja mudança nas regras de negócio de algum cálculo do sistema, apenas é necessária atualização do conteúdo da camada Control. Além do modelo arquitetural base, alguns padrões de projeto como Singleton (utilizado no Security Framework e DAO's) e Facade (agregando funções de diversas interfaces a uma interface centralizada que é exposta para os subsistemas), descritos por Gamma [5], foram aplicados pois o uso de padrões de projeto fornece soluções arquiteturais otimizadas para problemas conhecidos na engenharia de software. A Figura 12 mostra o diagrama de classes obtido.

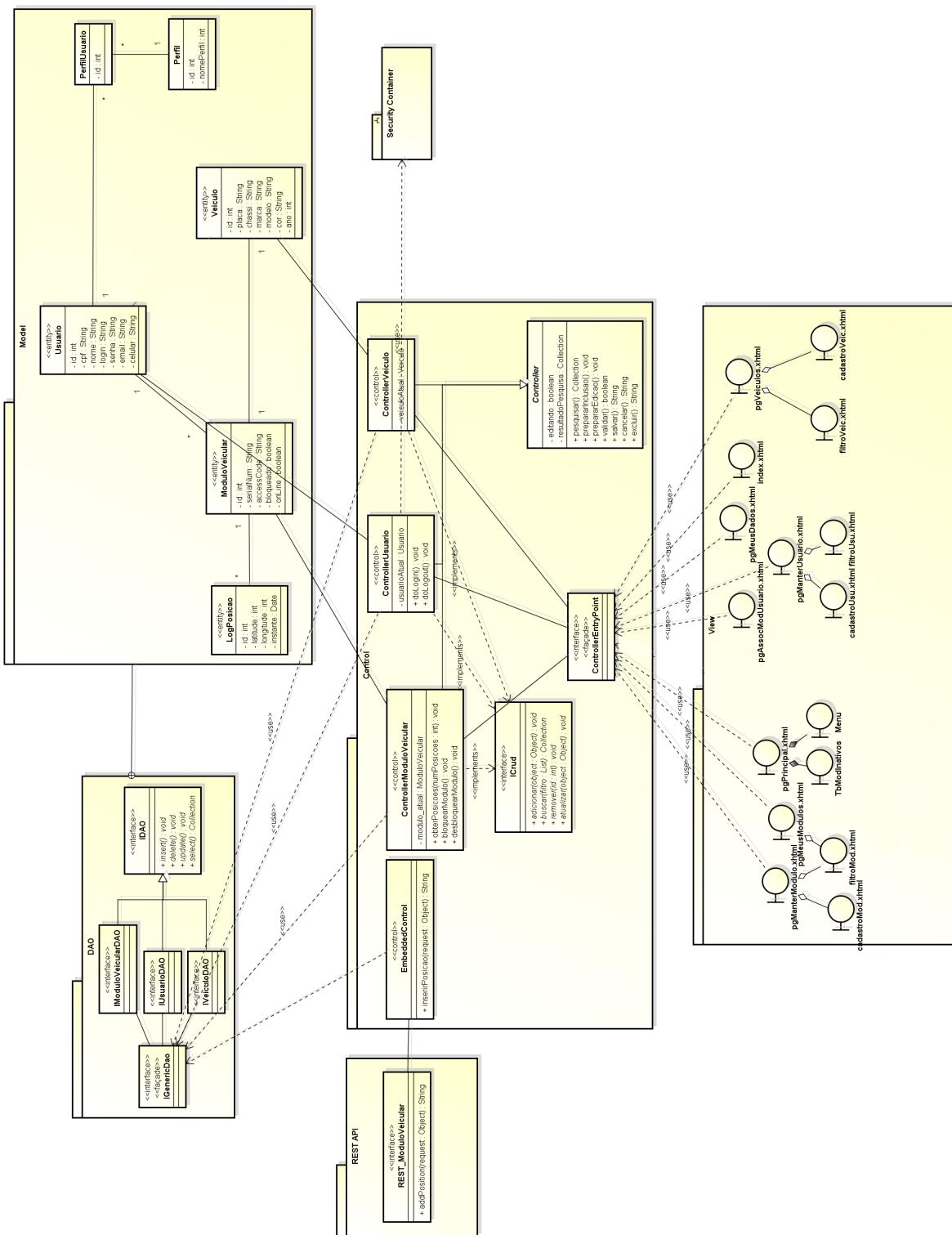


Figura 3.12: Modelo de classes do WebManager

3.8 Módulo Web (WebManager) - Desenvolvimento

A implementação em do código do WebManager contemplou primeiramente o web-service e as camadas de model e controllers, deixando a implementação da interface com o usuário como última etapa. A modelagem de banco de dados utilizando modelo entidade-relacionamento não foi necessária, uma vez que a camada Model contendo as entidades do sistema sofreu um mapeamento objeto-reacional (ORM) feito pelo framework Hibernate. Este mapeamento isenta o desenvolvedor de criar na maioria dos casos, cláusulas SQL, deixando-o apenas com o trabalho de manipular instâncias de classes e seus atributos. A camada de entidades é mostrada na figura 13.

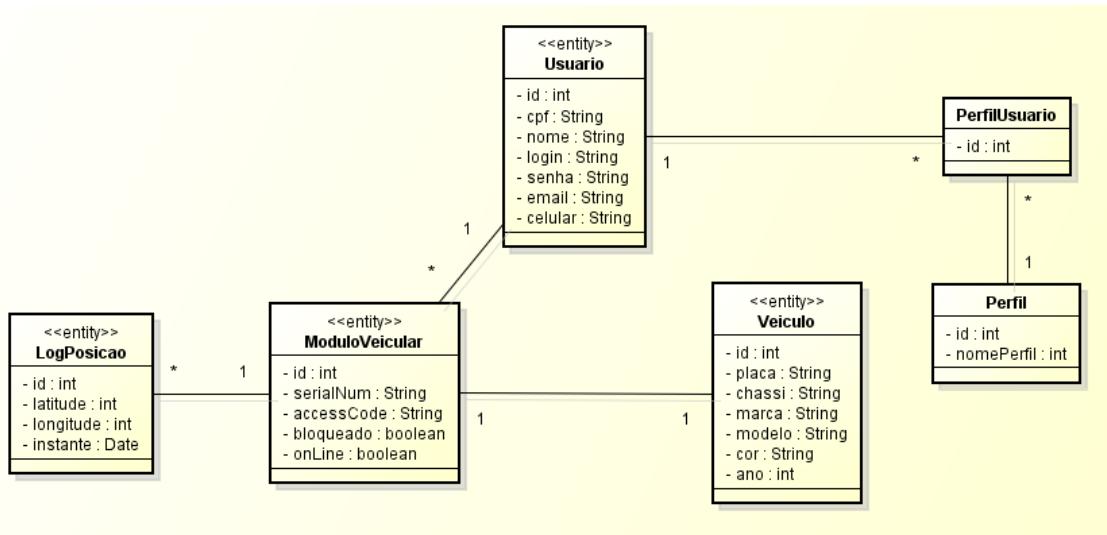


Figura 3.13: Model - Entidades do sistema

O funcionamento do módulo web é bastante simples, um usuário acessa o sistema por meio de um login e senha fornecidos, em caso de sucesso é direcionado à página de rastreamento onde pode efetuar a seleção do veículo pertencente a ele a fim de realizar o rastreamento, ação esta que é exibida em um mapa da API Google Maps. A figura 14 mostra a tela de login do sistema e a figura 15 exibe a tela de rastreamento principal.

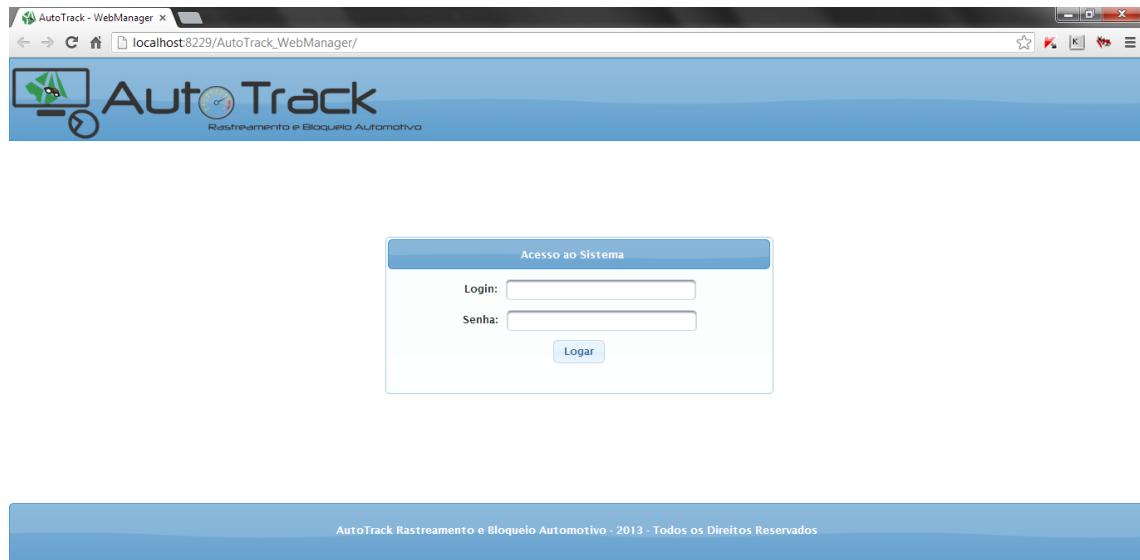


Figura 3.14: Tela de Login do Sistema

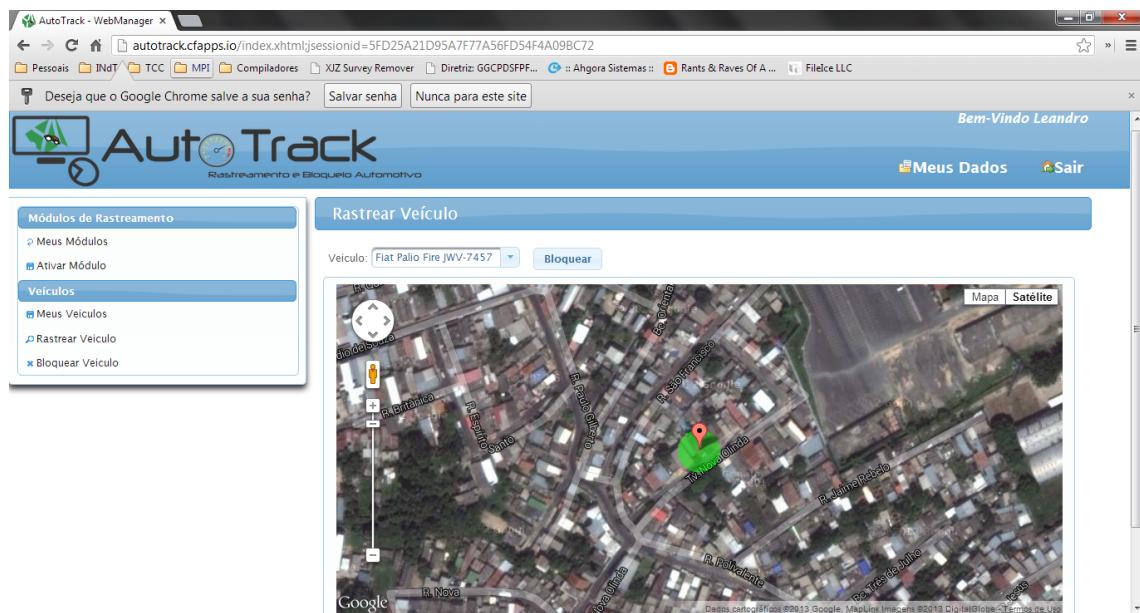


Figura 3.15: Tela de Rastreamento

Capítulo 4

Resultados e Discussões

4.1 Módulo Web (WebManager) - Testes de Carga

A fim de validar o comportamento do aplicação Web durante um alto número de requisições, caso que ocorre quando muitos módulos embarcados enviam requests HTTP no mesmo espaço de tempo (simulando uma situação de crescimento da plataforma), a ferramenta para testes de performance Apache JMeter foi utilizada para gerar o tráfego e o software VisualVM serviu para coletar métricas da aplicação como consumo de memória e uso da CPU.

Uma quantidade de 200 módulos concorrendo aos recursos do serviro foi aplicada, gerando resultados satisfatórios, com o consumo de memória em picos de 750 MB com gráfico de comportamento serrilhado, ou seja, liberação de memória não utilizada e ausência de leaks, além disso o consumo máximo de CPU ficou em torno de 30 %, uma margem segura. A figura 16 mosta o comportamento serrilhado do gráfico de consumo de memória, demonstrando a ausência de leaks e atuação correta do Garbage Collector.

O consumo de CPU mesmo com a ação do Garbage Collector ficou em margens coerentes de acordo com o gráfico da figura 17.

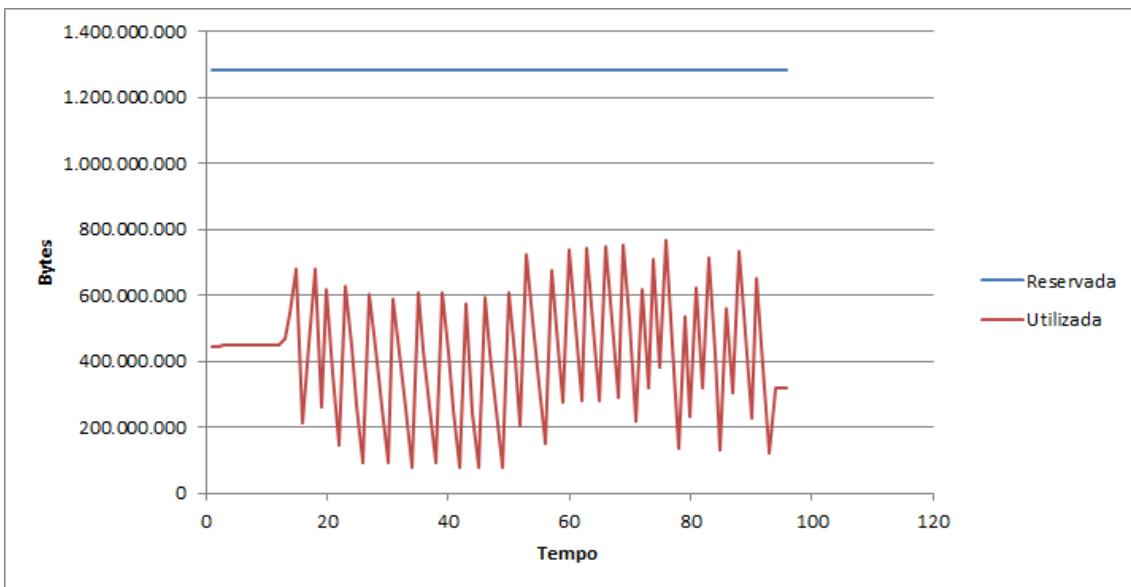


Figura 4.1: Gráfico do consumo de memória heap

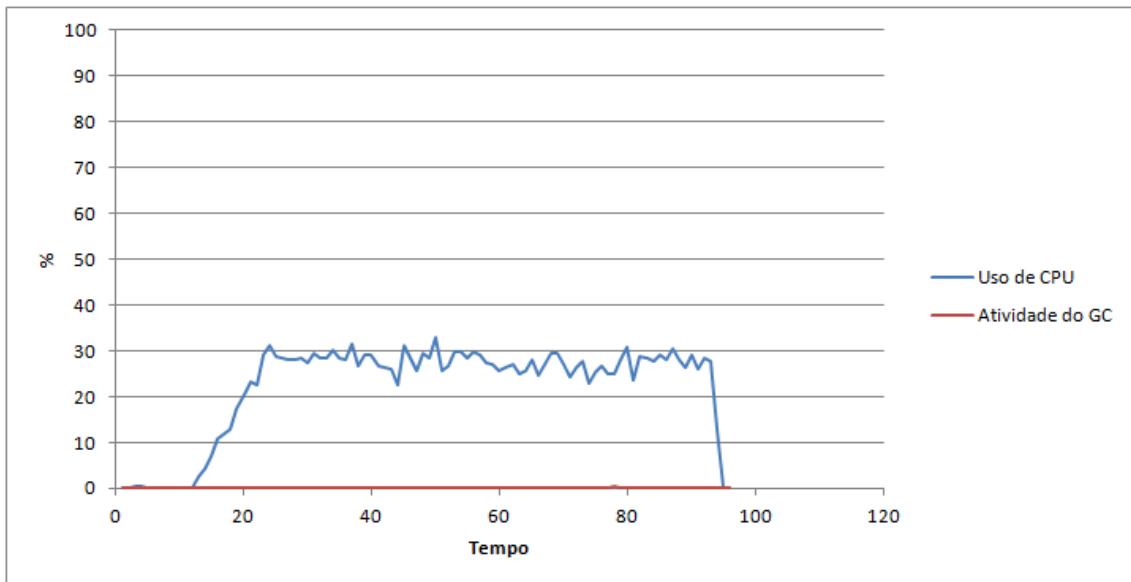


Figura 4.2: Gráfico do uso de CPU

4.2 Resultados

Os testes do protótipo incluem aplicação do módulo veicular em um automóvel para validar seu comportamento. Foi utilizado um veículo da marca Fiat modelo Palio Fire ano 2002, no qual o módulo embarcado foi acoplado no circuito de alimentação da bomba

de combustível para efetuar o corte de alimentação no caso da solicitação de bloqueio, resultando no desligamento do veículo. Os testes de rastreamento com automotor em movimento, bloqueio e desbloqueio foram realizados com sucesso e registrados em vídeo. A figura 18 mostra o protótipo integrado ao circuito elétrico do automóvel de testes.

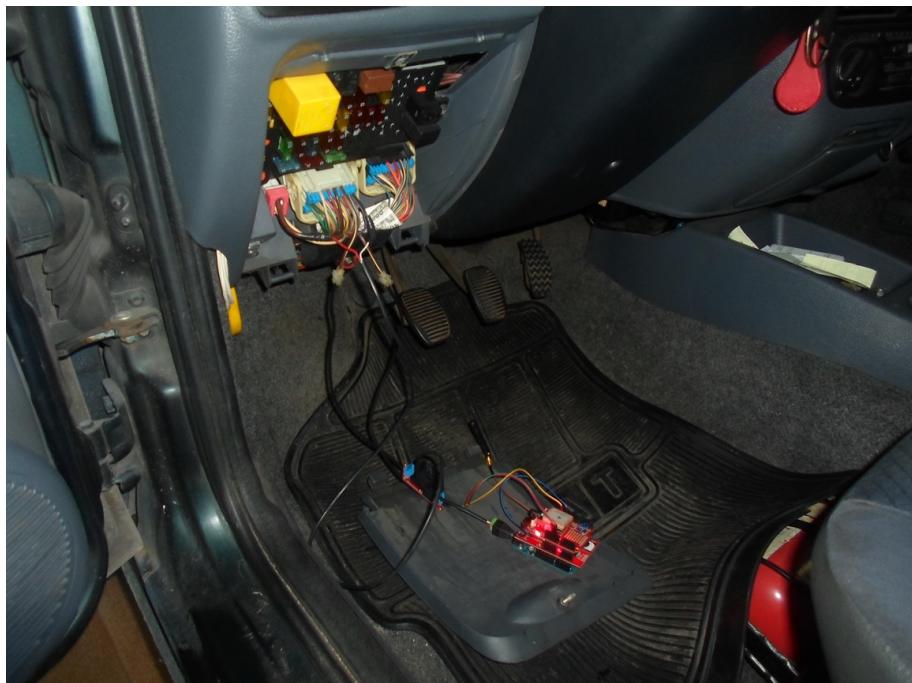


Figura 4.3: Integração do protótipo

Procedeu-se a coleta de dados que foram armazenados em banco relacional. As seções de coleta mostram um espaço aproximado de um minuto entre as inserções na base de dados, tempo previsto pois os parâmetros de delay setados no firmware do módulo embarcado foram calculados para resultar neste intervalo. A figura 19 mostra as linhas inseridas no banco de dados durante o período de testes.

	[PK] serial	instante timestamp without time zone	latitude real	longitude real	modulodeorig integer
1	1	2013-06-11 00:34:57.866	-3.1046	-3.1046	1
2	2	2013-06-11 00:42:48.318	-3.1046	-3.1046	1
3	3	2013-06-11 00:59:14.716	-3.1046	-3.1046	1
4	4	2013-06-11 00:59:30.328	-3.1046	-3.1046	1
5	5	2013-06-11 01:03:06.81	-3.1046	-3.1046	1
6	6	2013-06-11 01:06:53.994	-3.1046	-3.1046	1
7	7	2013-06-11 01:31:02.809	-3.1046	-3.1046	1
8	8	2013-06-11 01:32:34.405	-3.1046	-3.1046	1
9	9	2013-06-11 01:34:07.139	-3.1046	-3.1046	1
10	10	2013-06-11 01:40:40.465	-3.1047	-59.9845	1
11	11	2013-06-11 01:42:12.2	-3.1046	-59.9845	1
12	12	2013-06-11 01:43:44.395	-3.1046	-59.9845	1
13	13	2013-06-11 01:45:17.467	-3.1046	-59.9844	1
14	14	2013-06-11 01:46:50.892	-3.1046	-59.9845	1
15	15	2013-06-11 01:48:22.807	-3.1046	-59.9844	1
16	16	2013-06-11 01:49:55.895	-3.1046	-59.9844	1
17	17	2013-06-11 02:02:12.444	-3.1046	-59.9844	1
18	18	2013-06-11 02:05:16.147	-3.1046	-59.9844	1
19	19	2013-06-11 02:07:30.079	-3.1046	-59.9844	1
20	20	2013-06-11 02:14:42.444	-3.1046	-59.9845	1
21	21	2013-06-11 13:35:41.678	-3.1046	-59.9844	1

Figura 4.4: Dados coletados

4.3 Próximos Passos

- Finalizar a implementação da UI.
- Verificar a possibilidade de implementação de caminho feito pelo veículo.
- Verificar possibilidade de comunicação assíncrona entre módulo e webmanager.
- Melhorar eficiência do uso do módulo GSM.

4.4 Algoritmos

```

1  #include <SoftwareSerial.h>
2  #include <SM5100B_GPRS.h>
3  #include <TinyGPS.h>
4  #include <EEPROM.h>
5  #include <avr/pgmspace.h>
6
7  //#define DEBUG_MESSAGES
8
9  #define MAX_NUM_ERRORS 10
10 #define GSM_TX_PIN 2
11 #define GSM_RX_PIN 3
12 #define GPS_TX_PIN 4
13 #define GPS_RX_PIN 5
14 #define GPS_T_OUT 5000
15
16 #define TIME_TO_SEND 5000
17 #define TIME_TO_READ_RESPONSE 2000
18
19 #define REBOOT_PIN 9
20 #define STATE_PERM_DATA_ADDR 0
21 #define STATE_UNBLOCKED LOW
22 #define STATE_BLOCKED HIGH
23 #define IO_PIN 12
24 #define ALARM_STATUS_PIN 10
25
26 #define BLOCK_MESSAGE "300"
27 #define UNBLOCK_MESSAGE "200"
28
29
30 TinyGPS gps;
31 SM5100B_GPRS cell(GSM_TX_PIN, GSM_RX_PIN);
32 SoftwareSerial gpsCommunicator(GPS_TX_PIN, GPS_RX_PIN);
33
34
35 String USER_AGENT = "Mozilla/5.0";
36 String HOST = "lm25ttt.no-ip.org";
37 int PORT = 8229;
38
39 String apn = "tim.br";
40 String user = "tim";
41 String password = "tim";
42 String path = "/AutoTrack_WebManager/api/embedded";
43 String responseFromServer = "";
44 byte pdpId = 1;
45 byte connectionId = 1;
46
47 byte numOfErrors=0;
48
49 byte moduleState = STATE_UNBLOCKED;
50
51 float actualLatitude = 0.0f;
52 float actualLongitude = 0.0f;
53
54 boolean attachNetwork()
55 {
56     if (cell.attachGPRS())
57     {
58 #ifdef DEBUG_MESSAGES
59         Serial.println(F("GPRS"));
60 #endif
61         if(cell.setUpPDPContext(&pdpId, &apn, &user, &password))
62         {
63 #ifdef DEBUG_MESSAGES
64             Serial.println(F("SetPDP"));
65 #endif
66             if(cell.activatePDPContext(&pdpId))
67             {
68 #ifdef DEBUG_MESSAGES
69                 Serial.println(F("ActivePDP"));
70 #endif
71                 return (true);
72             }
73         }
74     }
75     return (false);
76 }
```

```

1  boolean createSocket()
2  {
3      if(cell.connectToHostTCP(&connectionId, &HOST, &PORT))
4      {
5          #ifdef DEBUG_MESSAGES
6              Serial.println(F("Connect"));
7          #endif
8          if(cell.configureDisplayFormat(&connectionId, GSM_SHOW_ASCII, GSM_NOT_ECHO_RESPONSE))
9          {
10             #ifdef DEBUG_MESSAGES
11                 Serial.println(F("Display"));
12             #endif
13             return (true);
14         }
15     }
16 }
17 return false;
18 }

19
20
21 void rebootGSMPprocedure()
22 {
23     digitalWrite(REBOOT_PIN, LOW);
24 }
25
26 void signalizeError()
27 {
28     numErrors++;
29     if(numErrors>MAX_NUM_ERRORS)
30     {
31         rebootGSMPprocedure();
32     }
33 }
34
35 boolean sendMessageToServer(String *request, byte *connectionId)
36 {
37     if(cell.checkSocketStatusTCP())
38     {
39         numErrors=0;
40         #ifdef DEBUG_MESSAGES
41             Serial.println(F("Socket is Open"));
42         #endif
43         if(cell.sendData(request, connectionId))
44         {
45             #ifdef DEBUG_MESSAGES
46                 Serial.println(F("SendData"));
47             #endif
48             delay(TIME_TO_READ_RESPONSE);
49             responseFromServer = cell.getServerResponse(connectionId);
50             cell.cleanCounters();
51             return (true);
52         }
53         else
54         {
55             signalizeError();
56             #ifdef DEBUG_MESSAGES
57                 Serial.println(F("FAIL!!! SendData"));
58             #endif
59             return (false);
60         }
61     }
62     else
63     {
64         #ifdef DEBUG_MESSAGES
65             Serial.println(F("Fail on Socket Status!"));
66         #endif
67         while(!cell.dataStart(connectionId))
68         {
69             signalizeError();
70         }
71         signalizeError();
72         cell.cleanCounters();
73         return (false);
74     }
75     delay(100);
76 }

```

```

1  boolean doPost(byte *connectionId, String *path, float *latitude, float *longitude)
2  {
3      String request = "";
4      String parameters = buildJsonContent(latitude, longitude);
5      request += "POST ";
6      request += *path;
7      request += " HTTP/1.1\r\nHost: ";
8      request += HOST;
9      request += "\r\nConnection: keep-alive";
10     request += "\r\nUser-Agent: ";
11     request += (USER_AGENT+"\r\n");
12     request += "Accept: application/json\r\n";
13     request += "Content-Length: ";
14     request += parameters.length();
15     request += "\r\n";
16     request += "Content-Type: application/json\r\n\r\n";
17     request += parameters;
18
19     return sendMessageToServer(&request, connectionId);
20 }
21
22 String buildJsonContent(float *latitude, float *longitude)
23 {
24     String jsonContent= "";
25     jsonContent = "{";
26     jsonContent += "\"idModule\":\"12345678\", \"codAccess\":\"25897\", ";
27
28     char latConverted[10] = "";
29
30     jsonContent += "\"latitude\":=\"";
31     dtostrf(*latitude, 1, 4, latConverted);
32     jsonContent+= latConverted;
33
34     char longConverted[10] = "";
35
36     jsonContent += "\",\"longitude\":=\"";
37     dtostrf(*longitude, 1, 4, longConverted);
38     jsonContent += longConverted;
39     jsonContent += "\",\"alarm\":=\"";
40     jsonContent += digitalRead(ALARM_STATUS_PIN);
41     jsonContent += "\"}";
42
43     return jsonContent;
44 }
45
46 static bool feedGps()
47 {
48     unsigned long checker = millis();
49     while (true)
50     {
51         if (gpsCommunicator.available() && gps.encode(gpsCommunicator.read()))
52             return true;
53         if((millis()-checker)>GPS_T_OUT)
54             return false;
55     }
56 }
57
58 void setup()
59 {
60     digitalWrite(REBOOT_PIN, HIGH);
61     pinMode(REBOOT_PIN, OUTPUT);
62
63     pinMode(IO_PIN, OUTPUT);
64     moduleState = EEPROM.read(STATE_PERM_DATA_ADDR);
65     digitalWrite(IO_PIN, moduleState);
66
67     pinMode(ALARM_STATUS_PIN, INPUT);
68 #ifdef DEBUG_MESSAGES
69     Serial.begin(9600);
70 #endif
71     cell.initializeModule(9600);
72     attachNetwork();
73     createSocket();
74     gpsCommunicator.begin(4800);
75 #ifdef DEBUG_MESSAGES
76     Serial.println(F("Setup Ok!"));
77 #endif
78 }
79

```

```

1
2
3 void treatServerResponse(String *response)
4 {
5     if (response->substring(17, 20)==BLOCK_MESSAGE)
6     {
7         if(moduleState!=STATE_BLOCKED)
8         {
9             moduleState=STATE_BLOCKED;
10            EEPROM.write(STATE_PERM_DATA_ADDR, moduleState);
11            digitalWrite(IO_PIN, moduleState);
12        }
13    }
14    if(response->substring(17, 20)==UNBLOCK_MESSAGE)
15    {
16        if(moduleState!=STATE_UNBLOCKED)
17        {
18            moduleState=STATE_UNBLOCKED;
19            EEPROM.write(STATE_PERM_DATA_ADDR, STATE_UNBLOCKED);
20            digitalWrite(IO_PIN, moduleState);
21        }
22    }
23 }
24
25 void loop()
26 {
27     gpsCommunicator.listen();
28     delay(5000);
29
30     if(feedGps())
31         gps.f_get_position(&actualLatitude, &actualLongitude);
32
33     if((actualLatitude==0.0f) || (actualLongitude==0.0f))
34         return;
35
36     cell.listen();
37     delay(5000);
38
39     if(doPost(&connectionId, &path, &actualLatitude, &actualLongitude))
40     {
41         treatServerResponse(&responseFromServer);
42 #ifdef DEBUG_MESSAGES
43         Serial.println(F("Delay"));
44 #endif
45         delay(TIME_TO_SEND);
46     }
47 #ifdef DEBUG_MESSAGES
48     Serial.println(responseFromServer);
49 #endif
50     responseFromServer = "";
51 }
52

```

Código 4.4.4: Código do Módulo Embarcado Parte 4

Bibliografia

- [1] Paulo Albuquerque e Cáudia Santos. *GPS para iniciantes*. Apostila do Mini Curso de GPS - XI Simpósio Brasileiro de Sensoriamento. Abr. de 2003.
- [2] Arduino. *Arduino Uno Specifications*. Set. de 2013. ENDEREÇO: <http://arduino.cc/en/Main/ArduinoBoardUno> (acesso em 12/09/2013).
- [3] Rafael Borges. *Furtos e roubos de carros avançam em grandes capitais do Brasil*. Maio de 2011. ENDEREÇO: <http://noticias.uol.com.br/cotidiano/ultimas-noticias/2011/05/08/furtos-e-roubos-de-carros-avanca-em-grandes-capitais-do-brasil-e-assusta-moradores.htm> (acesso em 12/02/2013).
- [4] Huascar Espinoza, David Servat e Sébastien Gérard. “Leveraging analysis-aided design decision knowledge in UML-based development of embedded systems”. Em: *Proceedings of the 3rd international workshop on Sharing and reusing architectural knowledge*. SHARK '08. Leipzig, Germany: ACM, 2008, pp. 55–62. ISBN: 978-1-60558-038-8. DOI: 10.1145/1370062.1370078. ENDEREÇO: <http://doi.acm.org/10.1145/1370062.1370078>.
- [5] Erich Gamma. *Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos*. Bookman, 2000.
- [6] Arduino ICSP. *Arduino Uno ICSP*. Set. de 2013. ENDEREÇO: <http://arduino.cc/en/Hacking/Programmer> (acesso em 12/09/2013).
- [7] Alfonso Diaz-Granados Marquez. “Sistema Antifurto de Veículos Automotivos”. Graduação. Curitiba: Núcleo de Ciências Exatas e de Tecnologia, Centro Universitário Positivo, 2006, p. 34.
- [8] Grant Martin, Luciano Lavagno e Jean Louis-Guerin. “Embedded UML: a merger of real-time UML and co-design”. Em: *Proceedings of the ninth international symposium on Hardware/software codesign*. CODES '01. Copenhagen, Denmark: ACM, 2001, pp. 23–28. ISBN: 1-58113-364-2. DOI: 10.1145/371636.371660. ENDEREÇO: <http://doi.acm.org/10.1145/371636.371660>.
- [9] Leandro Borges Martins. “Sistema Antifurto Integrado ao Monitoramento de Presença de Crianças no Interior de veículos usando GPRS”. Graduação. Brasília: Faculdade de Tecnologia e Ciências Sociais Aplicadas, Centro Universitário de Brasília, 2010, p. 65.
- [10] Peter Marwedel. *Embedded System Design: Embedded Systems Foundations of Cyber-Physical*. 2nd Edition. Springer, 2001.
- [11] Antonio Mendes. *Arquitetura de Software: Desenvolvimento orientado para arquitetura*. Editora Campus, 2002.
- [12] Ivan Sampaio Nascimento. “Sistema de alarme automotivo que integra transdutor acústico/elétrico e celular”. Graduação. Brasília: Faculdade de Ciências Exatas e Tecnologia, Centro Universitário de Brasília, 2007, p. 54.

- [13] Sparkfun GPS Shield. *Retail Kit*. Abr. de 2013. ENDEREÇO: <https://www.sparkfun.com/products/10709> (acesso em 10/04/2013).
- [14] Wayne Wolf. *Computer as Components: Principles of Embedded Computing System Design*. 2nd Edition. Morgan Kaufmann Publishers, 2001.