

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 BASEADO EM ARDUINO**

Manaus

2013

LEANDRO MAURÍCIO ARAÚJO BENTES

**SISTEMA DE SEGURANÇA VEICULAR COM USO DE GPS
BASEADO EM ARDUINO**

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:

Mário Augusto Bessa de Figueiredo

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 TCC:

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 baseado em Arduino /
Leandro Bentes; [orientado por] Prof. Dr. Raimundo Corrêa de Oliveira -
Manaus: UEA, 2013.

112 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
BASEADO EM ARDUINO

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
UNIVERSIDADE DO ESTADO DO AMAZONAS

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 poder 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, mesmo que tenhamos trilhado diferentes caminhos agradeço por poder dividir alegrias e angústias durante o tempo em que ficamos em na faculdade.

Resumo

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

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 | x |
| Lista de Figuras | xii |
| Lista de Códigos | xiii |
| 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 Referencial Tecnológico | 6 |
| 2.1 Sistema Embarcado | 6 |
| 2.2 Projeto de Sistemas Embarcados | 7 |
| 2.3 Microcontroladores | 10 |
| 2.4 Plataforma Arduino | 11 |
| 2.4.1 Arduino Shields | 12 |
| 2.5 Placa Arduino Uno | 14 |
| 2.5.1 Especificações Arduino Uno | 15 |
| 2.5.2 Alimentação da Placa Arduino Uno | 15 |
| 2.5.3 Entrada e Saída | 17 |
| 2.5.4 Programação | 18 |
| 2.6 Tecnologia GPS | 19 |
| 2.6.1 Modelo de Posicionamento GPS | 19 |
| 2.7 Arduino GPS Shield Kit | 21 |
| 2.8 Comunicação Móvel | 23 |
| 2.8.1 Tecnologia GPRS | 24 |

| | | |
|----------|--|----|
| 2.8.2 | Arduino Cellular Shield with SM5100B | 25 |
| 2.9 | Controle de carga via Arduino Uno | 26 |
| 2.10 | Webservice | 28 |
| 2.10.1 | REST | 29 |
| 2.10.2 | API Web baseada em REST | 30 |
| 2.11 | Padrões de Projeto | 31 |
| 2.11.1 | MVC | 31 |
| 2.11.2 | Singleton | 32 |
| 2.11.3 | Dao | 33 |
| 2.11.4 | Factory Method | 33 |
| 3 | Sistema de Segurança Veicular com uso de GPS baseado em Arduino 34 | |
| 3.1 | Levantamento de requisitos | 34 |
| 3.2 | Especificação de Requisitos | 35 |
| 3.2.1 | Casos de Uso | 36 |
| 3.2.2 | Descrição dos Casos de Uso | 36 |
| 3.3 | Modelo de classes para o módulo embarcado | 40 |
| 3.4 | Arquitetura do Sistema | 42 |
| 3.5 | Hardware do módulo embarcado | 44 |
| 3.5.1 | Placa controladora | 45 |
| 3.5.2 | Shield GSM | 46 |
| 3.5.3 | Shield GPS | 47 |
| 3.5.4 | Circuito de I/O com Veículo | 47 |
| 3.6 | Desenvolvimento do Firmware | 51 |
| 3.6.1 | Interface com EEPROM | 53 |
| 3.6.2 | Interface com I/O | 54 |
| 3.6.3 | Interface com Shield GPS | 55 |
| 3.6.4 | Interface com Shield GSM | 57 |
| 3.6.5 | Requisição ao webservice | 57 |
| 3.6.6 | Visão geral do funcionamento do módulo embarcado | 59 |
| 3.7 | Módulo Web (WebManager) | 62 |
| 3.7.1 | Modelo geral | 62 |
| 3.7.2 | Camada de modelo | 63 |
| 3.7.3 | Camada de controle | 66 |
| 3.7.4 | Webservice | 67 |
| 3.7.5 | View | 68 |

| | |
|---|-----------|
| 4 Resultados e Discussões | 71 |
| 4.1 Testes de Carga do Módulo Web | 71 |
| 4.2 Integração ao veículo | 73 |
| 4.3 Resultados | 74 |
| 5 Considerações Finais | 76 |
| 5.1 Conclusão | 76 |
| 5.2 Dificuldades encontradas | 77 |
| 5.3 Trabalhos futuros | 77 |
| 5.4 Disciplinas aplicadas | 78 |
| A Anexos | 82 |
| A.1 Descrição dos casos de uso | 82 |
| A.2 Códigos, esquemas e diagramas | 92 |

Listas de Tabelas

| | | |
|-----|---|----|
| 2.1 | Modelos de placa Arduino | 12 |
| 2.2 | Resumo das Especificações - Arduino Uno | 15 |
| 2.3 | Especificações operacionais do módulo GPS EM-406A | 22 |
| 2.4 | Descrições das seções da mensagem NMEA 0183 | 23 |
| 2.5 | Web API RESTful | 31 |
| 3.1 | Caso de Uso Obter Posição Atual | 37 |
| 3.2 | Configurar situação de Bloqueio | 39 |
| A.1 | Caso de Uso Manter Usuário | 82 |
| A.2 | Caso de Uso Manter ModVeicular | 84 |
| A.3 | Caso de Uso Associar Modulo-Usuário | 86 |
| A.4 | Caso de Uso Atualizar Próprio Cadastro | 87 |
| A.5 | Caso de Uso Manter Veículos | 88 |
| A.6 | Caso de Uso Listar Módulos | 90 |

Lista de Abreviaturas e Siglas

GPRS General Packet Radio Service

GPS Global Positioning System

GSM Global System for Mobile

NMEA National Marine Electronics Association

PWM Pulse-Width Modulation

UML Unified Modeling Language

Listas de Figuras

| | | |
|------|---|----|
| 2.1 | Exemplo de sistema embarcado: roteador wireless | 7 |
| 2.2 | Microcontrolador Atmel | 11 |
| 2.3 | Ilustração da placa Arduino com shields conectados | 14 |
| 2.4 | Placa Arduino Uno em detalhes | 15 |
| 2.5 | Fonte de alimentação para testes em bancada | 16 |
| 2.6 | Ilustração do ambiente de programação Arduino | 18 |
| 2.7 | Modelo conceitual da constelação de satélites GPS | 19 |
| 2.8 | Coordenada de um ponto pela triangularização do sinal de satélites GPS . | 20 |
| 2.9 | Elementos do sistema GPS | 21 |
| 2.10 | Kit GPS | 21 |
| 2.11 | Mensagem NMEA 0183 | 23 |
| 2.12 | Comutação de circuito e de pacote nas redes GSM e GPRS | 24 |
| 2.13 | Ilustração do Cellular Shield | 26 |
| 2.14 | Antena quad-band com 1.5 dBi de ganho | 26 |
| 2.15 | Circuito de controle de cargas com relê | 28 |
| 2.16 | Placa para controle de cargas por relê | 28 |
| 3.1 | Visão geral do sistema de segurança veicular com uso de GPS baseado em Arduino | 35 |
| 3.2 | Casos de uso do sistema de segurança veicular com uso de GPS baseado em Arduino | 36 |
| 3.3 | Diagrama de Classes do Módulo Veicular | 41 |
| 3.4 | Diagrama de Arquitetura do Sistema de Segurança | 42 |
| 3.5 | Exemplar Arduino Uno | 45 |
| 3.6 | Pinos para empilhamento no Shield GSM | 46 |
| 3.7 | Shield GSM integrado ao Arduino Uno | 46 |
| 3.8 | Modificação no Shield GPS | 48 |
| 3.9 | Integração Arduino Uno, GSM e GPS | 48 |

| | | |
|------|---|----|
| 3.10 | Círcuito normal da bomba de combustível | 49 |
| 3.11 | Círcuito da bomba após intervenção | 49 |
| 3.12 | Fios interligados à malha normalmente fechada do relé | 50 |
| 3.13 | Conjunto de hardware do módulo embarcado montado | 50 |
| 3.14 | Sequência lógica do firmware | 52 |
| 3.15 | Classe de comunicação com SM5100B | 58 |
| 3.16 | Requisição HTTP feita pelo módulo veicular | 58 |
| 3.17 | Proceso de setup | 60 |
| 3.18 | Atividades durante o loop | 61 |
| 3.19 | Model - Entidades do sistema | 63 |
| 3.20 | Classes DAO | 65 |
| 3.21 | Controllers | 66 |
| 3.22 | Esquema do webservice | 67 |
| 3.23 | Tela de Login do Sistema | 69 |
| 3.24 | Tela de Rastreamento | 70 |
| 4.1 | Gráfico do consumo de memória heap | 72 |
| 4.2 | Gráfico do uso de CPU | 72 |
| 4.3 | Integração do protótipo | 73 |
| 4.4 | Dados coletados | 74 |
| A.1 | Esquema elétrico da placa Arduino Uno | 97 |
| A.2 | Modelo de classes do WebManager | 98 |
| A.3 | Esquema elétrico do GPS Shield | 99 |

Lista de Códigos

| | |
|--|----|
| 3.6.1 Rotinas de leitura e escrita em EEPROM | 54 |
| 3.6.2 Rotinas de acionamento do pino digital para controle do relé | 55 |
| 3.6.3 Rotinas de obter a posição geográfica via GPS | 56 |
| A.2.1 Código do Módulo Embarcado Parte 1 | 93 |
| A.2.2 Código do Módulo Embarcado Parte 2 | 94 |
| A.2.3 Código do Módulo Embarcado Parte 3 | 95 |
| A.2.4 Código do Módulo Embarcado Parte 4 | 96 |

Capítulo 1

Introdução

1.1 Descrição do Problema

De acordo dados do DENATRAN [6] 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. O texto sintetiza informações das secretarias estaduais de segurança pública e dos Detrans, mostrando as cidades do Rio de Janeiro, São Paulo e Salvador ocupando, respectivamente, os três primeiros lugares em número de roubos e furtos no país.

Ainda segundo o DENATRAN [6], 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. Uma parte também 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 de 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 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 [16] descreve um projeto de sistema de alarme automotivo cuja principal funcionalidade é a detecção de um evento sonoro no interior do veículo e a sinalização a um microcontrolador. O microcontrolador, 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 de Nascimento [16] verificou-se uma abordagem bastante prática, com experimentos, construção de circuitos, exibição do funcionamento do protótipo e 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 [13] 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 for acionado, detectar quando algum objeto, criança ou animal de estimação for esquecido sobre o banco verificando a presença do peso sobre este.

O projeto se baseia na plataforma Arduino, utilizando a filosofia de hardware livre, modularização de componentes e desenvolvimento simplificado de protótipos para posterior fabricação, 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 [10] se refere à criação de um sistema de autenticação em que 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 radiofrequência, 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, o trabalho não traz modelagem de requisitos e de projeto baseada em UML, apenas diagramas resultantes da fase de implementação, além disso, o projeto não implementa uma interface amigável.

1.3 Contribuições

Este trabalho apresenta o desenvolvimento de um sistema de segurança veicular com uso de GPS baseado em Arduino, 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 [19] voltada a sistemas embarcados em que aplicam-se 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 o 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 que se encontra anexado a monografia em mídia digital.

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 apresenta os resultados obtidos com a implementação, dados de testes realizados durante o processo de desenvolvimento.
- O capítulo 5 descreve as conclusões e trabalhos futuros.

Capítulo 2

Referencial Tecnológico

2.1 Sistema Embarcado

Segundo Wolf [19], 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 se 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 tal maneira que o consumo de recursos energéticos seja sempre o mínimo possível. Um

elevado consumo influi 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 TCP/IP;
- Opera em tempo real: todos os pacotes que passam pelo roteador deve ser processados em tempo hábil;
- Custo de produção reduzido: 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.



Figura 2.1: Exemplo de sistema embarcado: roteador wireless

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 proposto por Martin et al. [11], que é um profile UML representando 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 [19], 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 enquanto se define um padrão UML para embarcados.

Em um segundo artigo, Martin [12] propõe novas extensões UML que cobrem lacunas não atendidas pela linguagem, como suporte a modelagem de hardware baseada em VHDL.

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 et al. [7] que propõem práticas complementares para analisar estes requisitos não funcionais, a fim de tornar os sistemas mais eficientes de confiáveis.

Wolf [19] propõe uma abstração em alto nível dos principais passos no projeto de sistemas embarcados em uma abordagem *top-down*, são estes:

- *Requisitos*: 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. Requisitos funcionais e não funcionais são levados em conta.
- *Especificação*: etapa em que se realiza o 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.

Marwedel [14] 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 encontrar o problema.
- *Testes de integração:* Verificação da comunicação entre os diferentes componentes quando 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 em relação ao descrito nas especificações, é um teste do sistema completo e em funcionamento, para confirmação se todos os requisitos 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 comparação com os microprocessadores, pode-se constatar que os microcontroladores operam em uma frequência muito baixa, porém 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 microcontroladores podem entrar em modo de espera, aguardando por um evento externo, tal como pressionar de uma tecla ou acionamento de um sensor.

Uma outra característica antagônica em relação a um microprocessador é que, enquanto neste último 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 componentes básicos ao seu funcionamento embutidos.

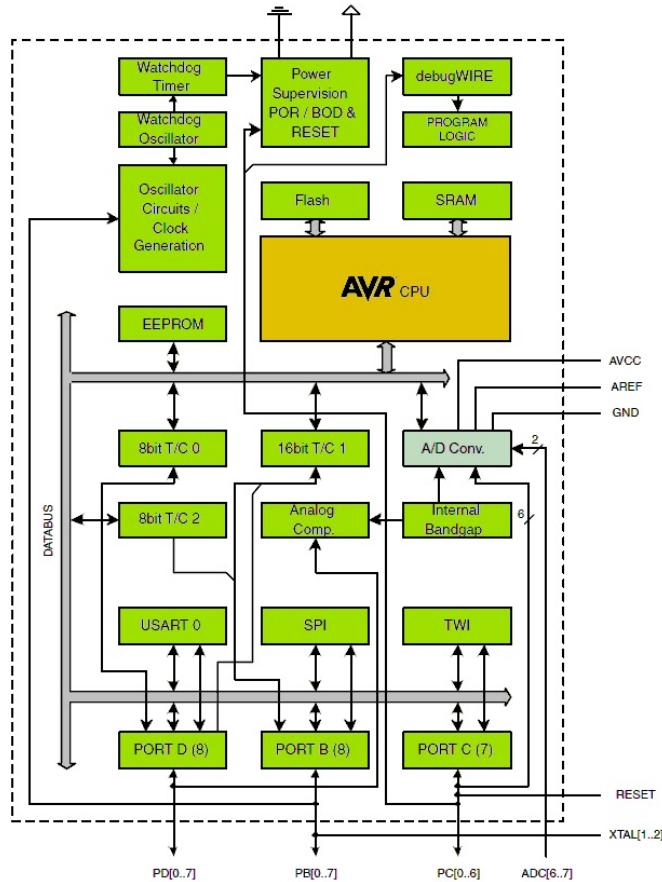


Figura 2.2: Microcontrolador Atmel

2.4 Plataforma Arduino

Arduino é uma plataforma para prototipagem eletrônica que utiliza o conceito de hardware livre. Neste conceito deve-se prover a disponibilização irrestrita de informações sobre o projeto de hardware, tais 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.

Os modelos mais utilizados da placa microcontroladora Arduino (Duemilanove e Uno) foram concebidas com base em um microcontrolador 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 microcontroladoras, cada uma carrega um codinome e especificações de hardware próprias. A tabela 2.1 descreve algumas placas da família Arduino.

Tabela 2.1: Modelos de placa Arduino

| Modelo | Clock | E/S Digital | E/S Analógico | Alimentação | Flash |
|---------------------|-------|-------------|---------------|-------------|--------|
| Arduino Due (ARM) | 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 |

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*.

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
- Bluetoooh
- 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.

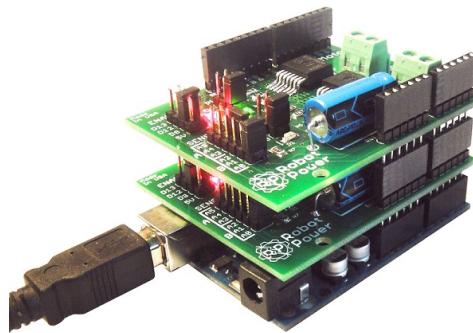


Figura 2.3: Ilustração da placa Arduino com shields conectados

2.5 Placa Arduino Uno

Para realizar a implementação deste projeto, optou-se por utilizar uma placa Arduino Uno pois projetos similares utilizaram esta placa sem problemas de sub ou super dimensionamento, ou seja, suas especificações atendem exatamente a necessidade do projeto, seu layout permite a conexão da maioria dos shields disponíveis e existe documentação disponível na Internet.

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 e botão de reset. 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 *bootloader*¹ que automatiza este passo (Arduino [3]).

A figura 2.4 exibe uma tomada frontal da placa Arduino Uno onde é visível o microcontrolador ATmega328 em encapsulamento SMD. 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. O esquemático elétrico da placa está ilustrado na figura ?? nos anexos.

¹*Bootloader* é o gerenciador de inicialização do microcontrolador.

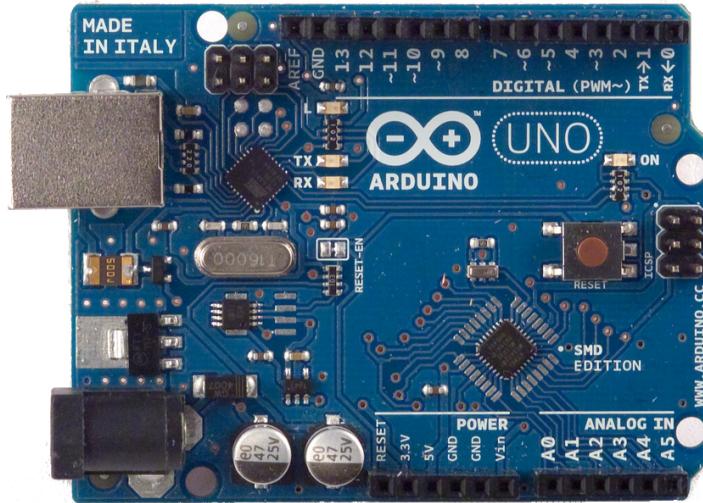


Figura 2.4: Placa Arduino Uno em detalhes

2.5.1 Especificações Arduino Uno

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

Tabela 2.2: Resumo das Especificações - 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 |

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 projeto foi utilizada, para testes em bancada, uma fonte regulada com saída de 12V e 1850mA de corrente máxima, ilustrada 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 tensão já que os veículos utilizam uma tensão 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. Permitem 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.
- LED: 13. Este pino possui um diodo emissor de luz acoplador permitindo testes de entrada/saída com facilidade.

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

2.5.4 Programação

A programação do microcontrolador ATmega328 do Arduino Uno pode ser feita através da IDE Arduino, ilustrada 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.

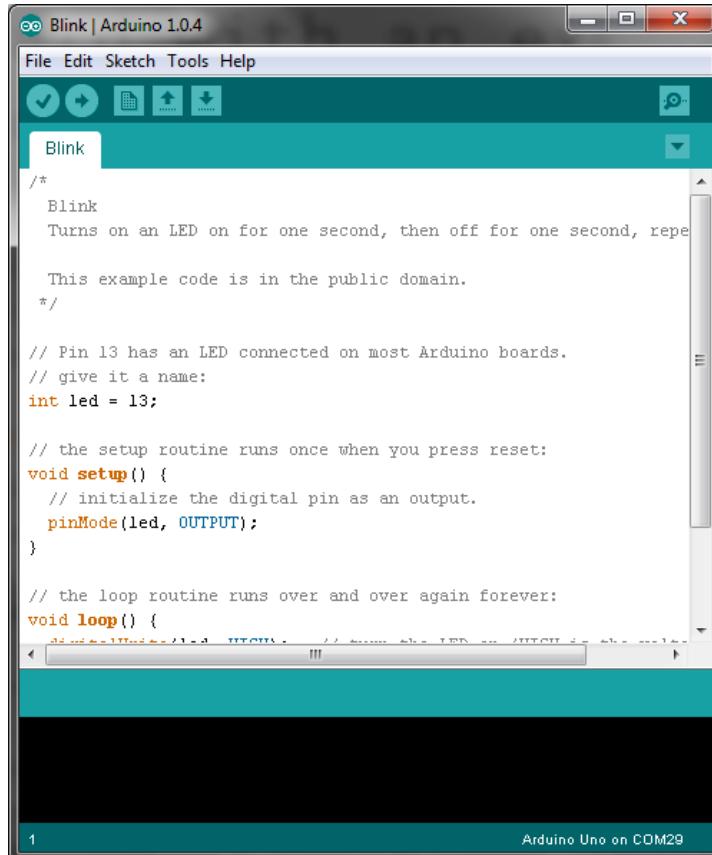


Figura 2.6: Ilustração do ambiente de programação Arduino

Sempre que é feita uma gravação de rotina no controlador ocorrerá uma reinicialização do circuito, forçando a sobrescrita 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. Em síntese, é 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 [2], 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. Com esta configuração fica garantido que no mínimo 4 satélites visíveis na superfície da Terra em qualquer horário do dia. A figura 2.7 ilustra uma idéia conceitual desta configuração.

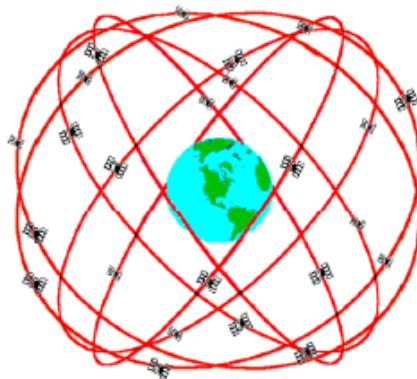


Figura 2.7: Modelo conceitual da constelação de satélites GPS

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 ilustra este processo, onde o ponto é calculado através das referências de três satélites e estimada quando existe sinal de apenas dois satélites.



Figura 2.8: Coordenada de um ponto pela triangularização do sinal de satélites 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 ao projeto, 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 [18]), que se mostrou financeiramente mais viável, além disso o fabricante disponibiliza todo arcabouço documental, programas de exemplo e esquemáticos de hardware. A figura 2.10 ilustra os elementos do kit GPS.

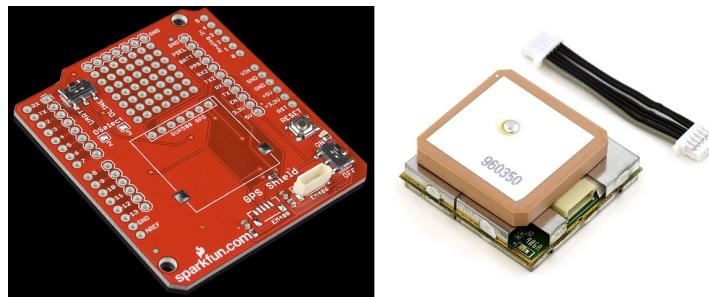


Figura 2.10: Kit GPS

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

- Shield (ou placa base): É uma placa de circuito impresso com layout padrão Ar-

duino 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 A.3 nos anexos ilustra o esquemático 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 apresenta os dados operacionais do módulo EM-406A.

Tabela 2.3: Especificações operacionais do módulo GPS 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 |

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 ilustrada na figura 2.11 e cada seção dela é descrita 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 2.11: Mensagem NMEA 0183

Tabela 2.4: Descrições das seções da 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 |

2.8 Comunicação Móvel

Segundo Guimarães [9], ”pode-se definir como comunicação móvel aquela onde existe a possibilidade de movimento relativo entre partes ou as partes sistêmicas envolvidas. Como exemplo temos a comunicação entre aeronaves, entre aeronaves e uma base terrena, entre veículos, a telefonia celular, a computação móvel, algumas classes de sistemas de telemetria e outros.”

Portanto, devido a característica de mobilidade deste trabalho, onde existe um elemento que se desloca no espaço e ao mesmo tempo necessita trocar dados com um sistema na Internet, este se classifica como um sistema de comunicação móvel. Para suprir as necessidades do projeto, os serviços de telefonia celular, mais especificamente serviços de internet móvel GPRS foram utilizados.

2.8.1 Tecnologia GPRS

O GPRS (General Packet Radio Service) é um serviço que foi criado para possibilitar o tráfego de dados por pacotes na rede GSM, possibilitando que a rede de telefonia celular possa ser ligada a Internet (3GPP [1]).

Na especificação inicial do GSM, a comunicação era realizada através de comutação de circuitos, onde uma conexão entre dois pontos de rede era alocada de forma a estar sempre disponível; a comunicação é feita de forma ininterrupta, uma otimização para chamadas de voz.

Porém a Internet é baseada na comutação de pacotes, e pela natureza de circuitos da rede GSM inicial, a velocidade de acesso era muito pequena. Para corrigir esta questão, a tecnologia GPRS foi criada e implementada se comunicando o GSM, trazendo a comutação de pacotes para a rede celular. Neste tipo de comutação, o ponto de origem envia uma informação dentro de um pacote que leva o endereço de destino em seu cabeçalho, o pacote então é transmitido pela rede que é responsável por escolher o melhor caminho até o ponto final.

Na figura 2.12 é ilustrada a comutação para dados de voz e dados por pacote. O elemento RTPC representa a rede para comunicação de voz, enquanto o elemento IP (Internet Protocol) representa os pontos de integração com a Internet.

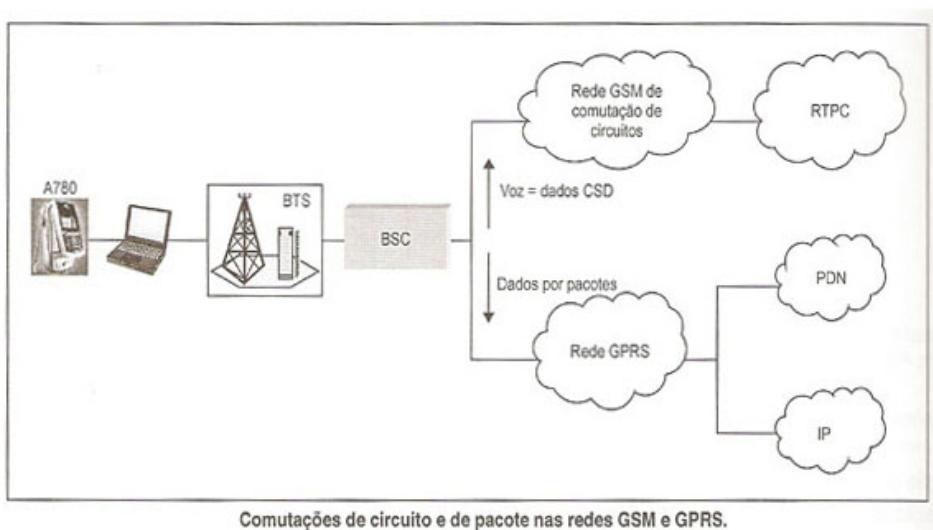


Figura 2.12: Comutação de circuito e de pacote nas redes GSM e GPRS

O endereçamento em GPRS é estabelecido em referência a o APN (Access Point

Name) da rede. O APN define os serviços de comunicação com a Internet como email e acesso a World Wide Web. Para iniciar uma conexão GPRS, o usuário deve especificar o APN, e opcionalmente um nome de usuário e uma senha, todos estes dados são fornecidos pela operadora.

2.8.2 Arduino Cellular Shield with SM5100B

Assim como a funcionalidade GPS, existe um kit shield que implementa o acesso ao serviço GPRS das operadoras de celular. Após pesquisa de referências, análise de preços e análise da facilidade de compra, optou-se pelo Cellular Shield com SM5100B do fabricante Sparkfun. O kit é composto pelos seguintes elementos:

- Placa de circuito impresso layout padrão Arduino Uno com circuito regulador de tensão, socket para cartão SIM e conector para antena.
- Módulo (ou modem) SM5100B da fabricante Spreadtrum, é um módulo miniaturizado, quad-band GSM 850/900/1800/1900 MHz de frequência de rede que implementa as funcionalidades de um celular como SMS, GSM/GPRS, TCP/IP e chamadas de Voz.
- Antena quad-band.

O módulo se comunica com o microcontrolador através de comandos AT transmitidos por um canal de comunicação serial bidirecional (são utilizados dois pinos digitais do Arduino Uno para montar este canal). A lista de comandos AT está descrita no manual fornecido pelo fabricante, bom como seus parâmetros e respostas. Para funcionamento do kit, é necessário um cartão SIM devidamente habilitado e com plano de dados ativo para as funções GPRS.

A figura 2.13 mostra o kit acoplado a placa Arduino Uno. Devido a necessidade de ganho de sinal, foi utilizada uma antena quad-band de 1.5 dBi similar a mostrada na figura 2.14.



Figura 2.13: Ilustração do Cellular Shield



Figura 2.14: Antena quad-band com 1.5 dBi de ganho

2.9 Controle de carga via Arduino Uno

Pela especificação da placa Arduino Uno observa-se que seus pinos digitais, quando configurados para saída, suportam apenas 40 mA de corrente máxima em 5 volts, isto é o suficiente para acionamento de leds, buzzers, sensores, ou seja, apenas cargas de pequeno porte.

Dependendo da natureza do projeto, pode ser necessário o acionamento de dispositivos que consumam corrente acima da saída máxima do microcontrolador e/ou operem em tensão diferente da plataforma. Para solucionar esta questão uma das alternativas é necessário utilizar um circuito auxiliar de chaveamento com relê, onde a

saída do microncontrolador é utilizada apenas como sinal para acionamento.

A figura 2.15 mostra o circuito de acionamento via relê (também conhecido como drive de acionamento), considerando que o relê do esquema possui uma bobina que opera com 5 volts, o funcionamento do circuito pode ser descrito com os seguintes passos:

- Os pontos descritos como RAW são alimentados com a saída de 5 volts do Arduino.
- No pino 2 do conector JP2 é ligada uma saída digital do microcontrolador e quando esta é acionada (5V), o transistor 2N3904 (utilizado como chave) permite a passagem de corrente polarizando inversamente o diodo D1.
- Com o diodo polarizado inversamente, a diferença de pontencial entre seus terminais se iguala a RAW, atingindo 5 volts.
- Nota-se que a bobina do relê está ligada em paralelo com o diodo D1, com isto a mesma é alimentada com a voltagem referente à polarização inversa do diodo, neste exemplo 5 volts. Como o relê opera a esta voltagem, a bobina se magnetiza chaveando os ponto LOAD1 e LOAD2 da figura.
- A corrente máxima que este circuito pode controlar depende da capacidade do relê. Valores de 5 a 20 ampéres são comuns.

Neste projeto foi utilizado uma placa pré-fabricada pelo fornecedor Sparkfun, conforme a figura 2.16 observa-se que o circuito conta com um pequeno LED sinalizador de estado e conectores para facilitar a instalação. O relê aplicado possui bobina que opera em 5 volts e possui duas malhas de chaveamento utilizáveis:

- Malha normalmente aberta (NO) que suporta corrente máxima de 20 ampéres em 240 volts.
- Malha normalmente fechada (NC) que suporta corrente máxima de 10 ampéres em 240 volts.

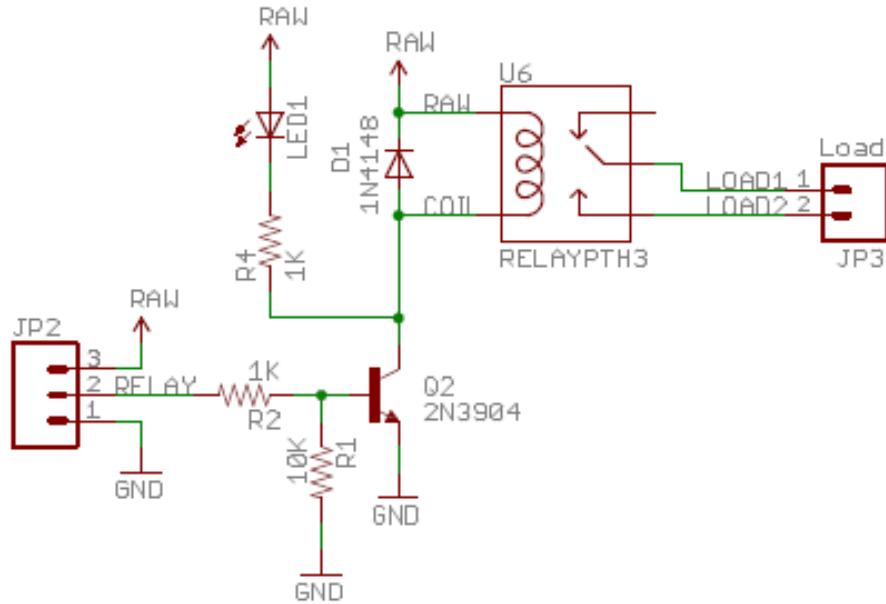


Figura 2.15: Circuito de controle de cargas com relê



Figura 2.16: Placa para controle de cargas por relê

2.10 Webservice

Um webservice é uma arquitetura de comunicação entre dispositivos sobre a Web. Pela definição da W3C é um sistema de software projetado para suportar a interoperabilidade de interação entre máquinas sobre uma rede, possuindo uma interface descrita em um formato processável (especificamente WSDL). Outros sistemas que interagem com um webservice tipicamente usando protocolo HTTP e dados estruturados representados

em XML em conjunto com outros tecnologias Web (Booth et al. [5]).

Essencialmente, um Webservice faz com que os recursos de uma aplicação estejam disponíveis sobre a rede de uma forma normalizada, onde cada recurso é identificado por um endereço universal único (URI) acessado pelo protocolo HTTP. Web Services é a tecnologia ideal para comunicação entre sistemas, sendo muito usado em aplicações Business to Business. A comunicação entre os serviços é padronizada possibilitando a independencia de plataforma e de linguagem de programação. Por exemplo, um sistema de reserva de passagens aéreas feito em Java e rodando em um servidor Linux pode acessar, com transparência, um serviço de reserva de hotel feito em .Net rodando em um servidor Microsoft.

O W3C define duas categorias de Webservice:

- REST-compliant Webservice: onde o principal propósito do serviço é manipular representações de recursos web em XML (ou similares) usando uma conjunto de operações sem estado.
- Webservices arbitrários: onde um webservice expõe um conjunto de operações por meio de uma linguagem de definição de webservices (WSDL).

2.10.1 REST

A Transferência de Estado Representacional (Representational State Transfer) ou somente (REST) é uma técnica de engenharia de software para sistemas hipermídia distribuídos como a Web. O termo se originou no ano de 2000, em uma tese de doutorado (PhD) sobre a web escrita por Roy Fielding, um dos principais autores da especificação do protocolo HTTP.

Na arquitetura REST se afirma que a web já fornece a escalabilidade como resultado de uma série de desenhos fundamentais:

- Um protocolo cliente/servidor sem estado: cada mensagem HTTP contém toda a informação necessária para compreender uma requisição, como resultado, nem o cliente e nem o servidor necessitam gravar nenhum estado das comunicações. Na prática, muitas aplicações baseadas em HTTP utilizam cookies e outros mecanismos para manter o estado da sessão.

- Um conjunto de operações bem definidas que se aplicam a todos os recursos de informação: HTTP em si define um pequeno conjunto de operações, as mais importantes são POST, GET, PUT e DELETE. Com frequência estas operações são combinadas com operações CRUD para a persistência de dados, onde POST não se encaixa exatamente neste esquema.
- Uma sintaxe universal para identificar os recursos. No sistema REST, cada recurso é unicamente direcionado através da sua URI.
- O uso de hipermídia (links) tanto para a informação da aplicação como para as transições de estado da aplicação: a representação deste estado em um sistema REST são tipicamente HTML ou XML. Como resultado disto, é possível navegar com um recurso REST a muitos outros, simplesmente seguindo ligações sem requerer o uso de registros ou outra infraestrutura adicional.

Um conceito importante em REST é a existência de recursos (elementos de informação), que podem ser usados utilizando um identificador global (um Identificador Uniforme de Recurso) para manipular estes recursos, os componentes da rede (clientes e servidores) se comunicam através de uma interface padrão (HTTP) e trocam representações de recursos (XML, JSON ou outro).

2.10.2 API Web baseada em REST

Segundo Booth et al. [5], uma API implementada utilizando os princípios do HTTP e REST é chamada Web API RESTful, sendo uma coleção de recursos com quatro aspectos principais:

- Uma URI base para a API como: `http://example.com/resources/`
- Um tipo de mídia para troca de dados, como XML, JSON ou qualquer outro tipo padronizado que segue o padrão hypertexto.
- Um conjunto de operações deve ser suportado usando os métodos HTTP (GET, PUT, POST ou DELETE). A API deve ser orientada a hypertexto.

A tabela 2.5 mostra como um serviço pode ser disponibilizado usando URIs e os métodos HTTP, sendo caracterizado como uma API RESTful.

Tabela 2.5: Web API RESTful

| Recurso | GET | PUT | POST | DELETE |
|---|--|--|---|--------------------------------|
| URI de coleções, como <i>http://example.com/resources</i> | Lista as URIs e outros detalhes dos membros da coleção. | Substitui a coleção inteira com outra coleção. | Cria uma nova entrada na coleção com URI designada automaticamente e retornada pela operação. | Deleta a coleção inteira. |
| URI de elemento, como <i>http://example.com/resources/item17</i> | Retorna uma representação do membro da coleção, expressada no formato apropriado (XML, JSON, etc). | Substitui o elemento do endereço ou, se não existir, o cria. | Geralmente não usado. Trata o elemento como uma coleção e cria uma nova entrada nele. | Deleta o elemento do endereço. |

2.11 Padrões de Projeto

Um Padrão de Projeto de Software, ou Design Pattern, descreve uma solução geral reutilizável para um problema recorrente no desenvolvimento de sistemas, com paradigma orientado a objetos (Gamma [8]). Não é um código final, é uma descrição ou modelo de como resolver o problema do qual trata, que pode ser usada em muitas situações diferentes. Os Padrões de Projeto normalmente definem as relações e interações entre as classes ou objetos, sem especificar os detalhes das classes ou objetos envolvidos, ou seja, estão num nível de generalidade mais alto.

Os padrões de projeto :

- Visam facilitar a reutilização de soluções de design, isto é, soluções na fase de projeto do software.
- Estabelecem um vocabulário comum de design, facilitando a comunicação, documentação e aprendizado dos sistemas de software.

2.11.1 MVC

Model-view-controller (MVC) é um modelo de arquitetura de software que separa a representação da informação da interação do usuário com ele. O modelo (model) consiste nos dados da aplicação, regras de negócios, lógica e funções. Uma visão (view) pode ser

qualquer saída de representação dos dados, como uma tabela ou um diagrama. É possível ter várias visões do mesmo dado, como um gráfico de barras para gerenciamento e uma visão tabular para contadores. O controlador (controller) faz a mediação da entrada, convertendo-a em comandos para o modelo ou visão. As ideias centrais por trás do MVC são a reusabilidade de código e separação de conceitos. (Reenskaug [17])

Além de dividir a aplicação em três tipos de componentes, o design MVC define as interações entre eles.

- Um controller pode enviar comandos para sua visão associada para alterar a apresentação da visão do modelo (por exemplo, percorrendo um documento). Ele também pode enviar comandos para o modelo para atualizar o estado do modelo (por exemplo, editando um documento).
- Um model notifica suas visões e controladores associados quando há uma mudança em seu estado. Esta notificação permite que as visões produzam saídas atualizadas e que os controladores alterem o conjunto de comandos disponíveis. Uma implementação passiva do MVC monta estas notificações, devido a aplicação não necessitar delas ou a plataforma de software não suportá-las.
- A visão (view) solicita ao modelo a informação que ela necessita para gerar uma representação de saída.

2.11.2 Singleton

Singleton é um padrão de projeto onde se garante a existência de apenas uma instância de uma classe, mantendo um ponto global de acesso ao seu objeto. Alguns projetos necessitam que algumas classes tenham apenas uma instância. Por exemplo, em uma aplicação que precisa de uma infraestrutura de log de dados, pode-se implementar uma classe no padrão singleton. Desta forma existe apenas um objeto responsável pelo log em toda a aplicação que é acessível unicamente através da classe singleton.

2.11.3 Dao

DAO (Data Access Object) é um padrão para persistência de dados que permite separar regras de negócio das regras de acesso a banco de dados. Numa aplicação que utilize a arquitetura MVC, todas as funcionalidades de bancos de dados, tais como obter as conexões, mapear objetos Java para tipos de dados SQL ou executar comandos SQL, devem ser feitas por classes de DAO.

2.11.4 Factory Method

Factory Method, é um padrão de projeto que permite as classes delegar para subclasses decidirem ações. O factory method permite delegar a instanciação para as subclasses. O padrão Factory Method contém os seguintes elementos:

- Creator — declara o factory method (método de fabricação) que retorna o objeto da classe abstrata. Este elemento também pode definir uma implementação básica que retorna um objeto de uma classe concreta básica;
- ConcreteCreator — sobrescreve o factory method e retorna um objeto da classe concreta;
- Product — define uma interface para os objetos criados pelo factory method;
- ConcreteClass — uma implementação para a interface.

Capítulo 3

Sistema de Segurança Veicular com uso de GPS baseado em Arduino

Este capítulo apresenta a implementação do Sistema de Segurança Veicular com uso de GPS baseado em Arduino contemplando a criação do conceito, etapa de modelagem, elaboração do hardware embarcado no veículo, finalizando com o desenvolvimento da aplicação Web responsável pela lógica de negócio e interface com usuário.

3.1 Levantamento de requisitos

Como ponto de partida é necessário definir o escopo do projeto partindo de uma idéia base, contendo os elementos interativos iniciais do sistema e sua representação conceitual. O cenário de operação deste projeto é composto por alguns elementos que emitem e consomem informação:

- Cada automóvel a ser verificado possui um módulo embarcado instalado e responsável por enviar sua localização geográfica atual, esta informação é obtida por meio da tecnologia GPS.
- Os dados obtidos são enviados a um serviço Web. Após pesquisa de trabalho similares e soluções aplicadas, foi decidido pelo uso da rede GPRS de uma operadora de telefonia móvel.

- O serviço Web captura as informações e armazena em uma base de dados.
- O usuário fará acesso ao sistema por meio de uma aplicação Web que oferece uma interface de acesso às principais operações, como localizar o veículo sobre uma API de mapas e opcionalmente realizar o desligamento remoto do mesmo.

A Figura 3.1 ilustra o modelo conceitual abstruído do sistema.

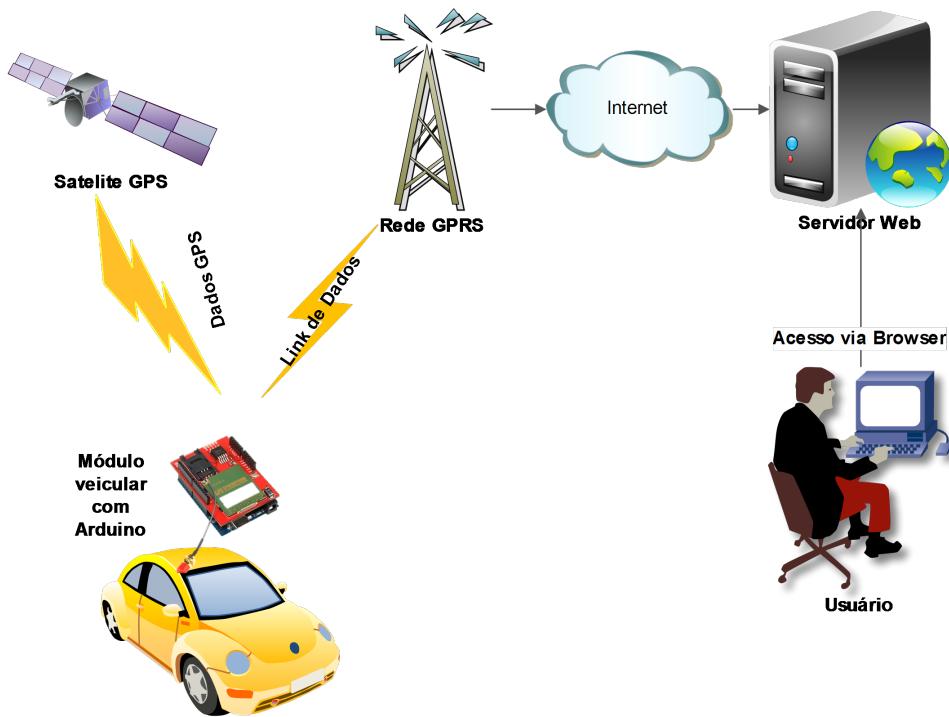


Figura 3.1: Visão geral do sistema de segurança veicular com uso de GPS baseado em Arduino

3.2 Especificação de Requisitos

Seguindo as recomendações de Wolf [19] para projeto de sistemas embarcados, é necessário capturar e descrever os requisitos. Nesta etapa, o Diagrama de Casos de Uso da UML foi utilizado, além disso a descrição dos casos permite o entendimento do fluxo de dados criando uma base para descrição dos componentes para implementação.

3.2.1 Casos de Uso

Foram identificados dois atores do sistema:

- Ator ”Usuário comum”: Faz o uso final do sistema, realizando atividades como rastreamento e bloqueio de veículos.
- Ator ”Administrador”: Realiza operações de gerenciamento de usuário e módulos veiculares.

A figura 3.2 ilustra o diagrama de casos de uso do projeto.

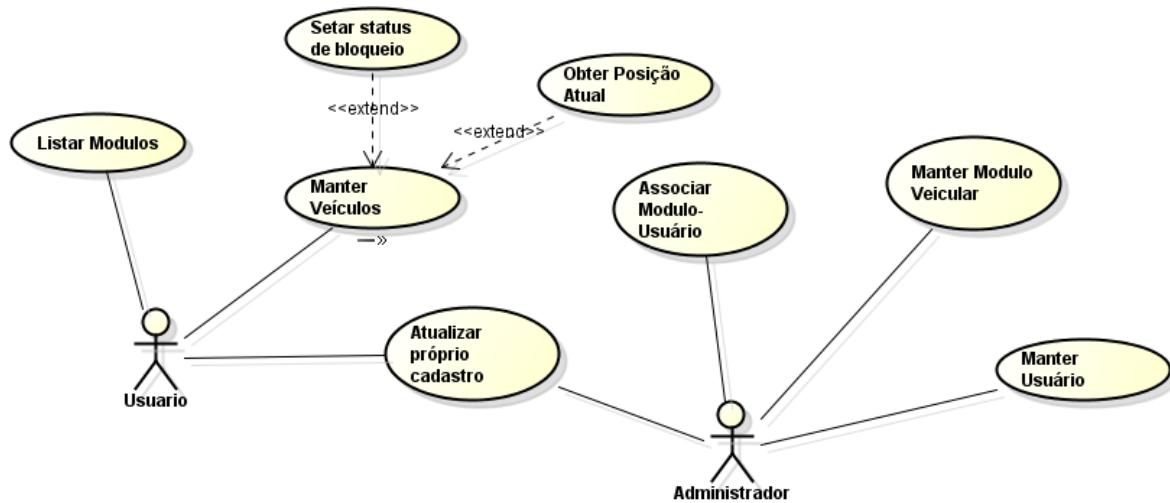


Figura 3.2: Casos de uso do sistema de segurança veicular com uso de GPS baseado em Arduino

3.2.2 Descrição dos Casos de Uso

Segundo Booch, Rumbaugh e Jacobson [4], podemos dizer que um caso de uso é um documento narrativo que descreve uma sequência de eventos de ator que usa um sistema para completar um processo, portanto, a descrição textual dos elementos da representação gráfica faz-se necessária.

As tabelas 3.1 e 3.2 descrevem os dois principais casos de uso do sistema, o restante das descrições encontra-se em anexo no final deste documento.

Tabela 3.1: Caso de Uso Obter Posição Atual

| | |
|---------------------|--|
| Nome do caso de uso | Obter posição atual |
| Sumário | Caso de uso que descreve a obtenção da posição atual de um veículo. |
| Ator primário | ”Usuário comum” |
| Atores secundários | |
| Precondições | Usuário cadastrado e equipamentos associados a ele. |
| Fluxo Principal | <ol style="list-style-type: none"> 1. Acessa página de rastreio. 2. Selecionar o veículo desejado. 3. O mapa é carregado na tela. 4. A posição atual é exibida no mapa, representada por um ponto especial. |
| Fluxo Alternativo | <p>(2) Erro na comunicação com o equipamento</p> <p>a. Em caso de falha na comunicação, o usuário é alertado com uma mensagem de indisponibilidade momentânea, porém incentivado a tentar novamente.</p> <p>(3) Erro no servidor de mapas</p> <p>a. O sistema deve informar o problema com os mapas, porém exibir os valores de longitude e latitude para que o usuário possa manualmente localizar o veículo.</p> <p>(4) Acionar função tipo de visão.</p> <p>a. Escolher entre visão de mapa ou visão de satélite.</p> |

| | |
|---------------------------|----------------|
| Pós-condições | |
| Requisitos não funcionais | |
| Autor | Leandro Bentes |
| Data | 15/04/2013 |

Tabela 3.2: Configurar situação de Bloqueio

| | |
|---------------------|---|
| Nome do caso de uso | Setar Status de Bloqueio |
| Sumário | Este caso de uso descreve como o usuário realiza o bloqueio ou desbloqueio do veículo através do sistema com o objetivo de facilitar a recuperação do automóvel. |
| Ator primário | “Usuário comum” |
| Atores secundários | |
| Precondições | Usuário cadastrado e equipamentos associados a ele. Veículo cadastrado e equipamento configurado. |
| Fluxo Principal | <ol style="list-style-type: none"> 1. Acessar página de rastreio. 2. Selecionar o veículo desejado. 3. Acionar botão bloquear. 4. O sistema exibe uma mensagem confirmando o bloqueio do veículo. |
| Fluxo Alternativo | <p>(2) Erro na comunicação com o equipamento</p> <p>a. Em caso de falha na comunicação, o usuário é alertado com uma mensagem de indisponibilidade momentânea, porém incentivado a tentar novamente.</p> <p>(2) Desbloquear Veículo</p> <p>a. Selecionar o botão de desbloqueio, que só estará disponível se houve um bloqueio anterior.</p> <p>b. O sistema exibe uma mensagem confirmando o desbloqueio do veículo.</p> <p>(5) Falha na ação</p> <p>a. O usuário é alertado com uma mensagem de indisponibilidade momentânea, porém incentivado a tentar novamente.</p> |

| | |
|---------------------------|----------------|
| Pós-condições | |
| Requisitos não funcionais | |
| Autor | Leandro Bentes |
| Data | 15/04/2013 |

3.3 Modelo de classes para o módulo embarcado

Para descrever os comportamentos de cada componente e integrá-los de forma lógica um Diagrama de Classes é desenhado, uma vez que a orientação a objetos atende de forma completa a descrição do conjunto de hardware. Pode-se abstrair comportamentos e representá-los através de métodos e abstrair estados como atributos de uma classe.

A principal vantagem em modelar os componentes de hardware com o diagrama 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.

Para exemplificar esta situação, neste projeto o componente GSM fornece uma gama de funcionalidade que não são utilizadas, como funções de chamada de voz, mensagem SMS e outras, porém como o modelo prevê apenas comportamentos relacionados a funções GPRS, não foi despendido tempo com a implementação de serviços para acessar estas funcionalidades.

A Figura 3.3 ilustra o diagrama de classes UML que abstrai as funcionalidades do módulo veicular embarcado e o representa de forma estruturada lógica.

Conforme se observa no desenho de classes, o módulo se resume a quatro classes, em que existe uma composição na classe principal ModuloVeicularHw, isto significa que esta só existe juntamente com as outras da composição e se uma destas últimas for removida a principal deixará de existir. Ainda na classe principal são guardados os atributos de estado, conexão e posição atual, além de alguns atributos para verificação de erros.

A classe GSM possui como atributo principal uma lista de comandos AT, utilizados para acionar as diversas funções de celular, estes comandos são strings definidas pelo fabricante e descritas no manual do produto. Nota-se que apenas métodos relacionados com a função GPRS e conexões de rede dados foram mapeados, pois só existe a necessidade de uso nesta modalidade.

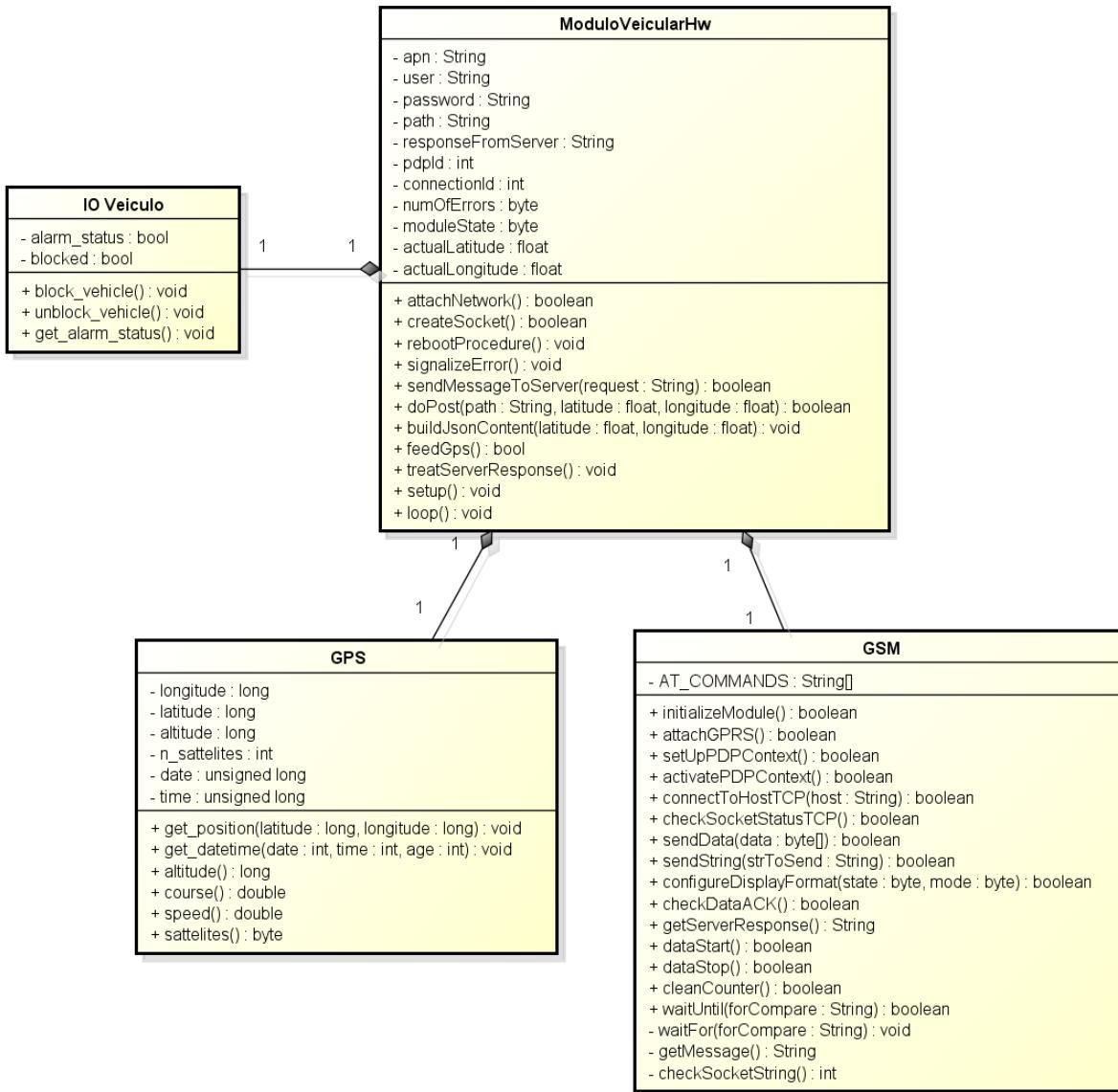


Figura 3.3: Diagrama de Classes do Módulo Veicular

No modelo da classe GPS, os atributos principais utilizados para obter posição geográfica atual são latitude e longitude, porém para o funcionamento do hardware dados de altitude, data, hora e número de satélites também são utilizados. Como comportamentos padrão existem: a obtenção da posição atual, velocidade, curso, satélites e data com hora.

O componente IO Veículo apenas representa as ações de bloquear, desbloquear e estado do alarme (não utilizado neste projeto). Como atributos, o estado do alarme e estado de bloqueio são representados.

3.4 Arquitetura do Sistema

Baseado no modelo conceitual, nos casos de uso, suas descrições e pesquisa de trabalhos correlatos, é possível estabelecer um modelo arquitetural transcrito em um diagrama de componentes UML, seguindo recomendações estabelecidas para sistemas de tempo real de acordo com Mendes [15]. Nesta etapa da modelagem são definidas algumas tecnologias utilizadas, uso de serviços externos e protocolos de comunicação. A Figura 3.4 mostra o diagrama de componentes da UML gerado.

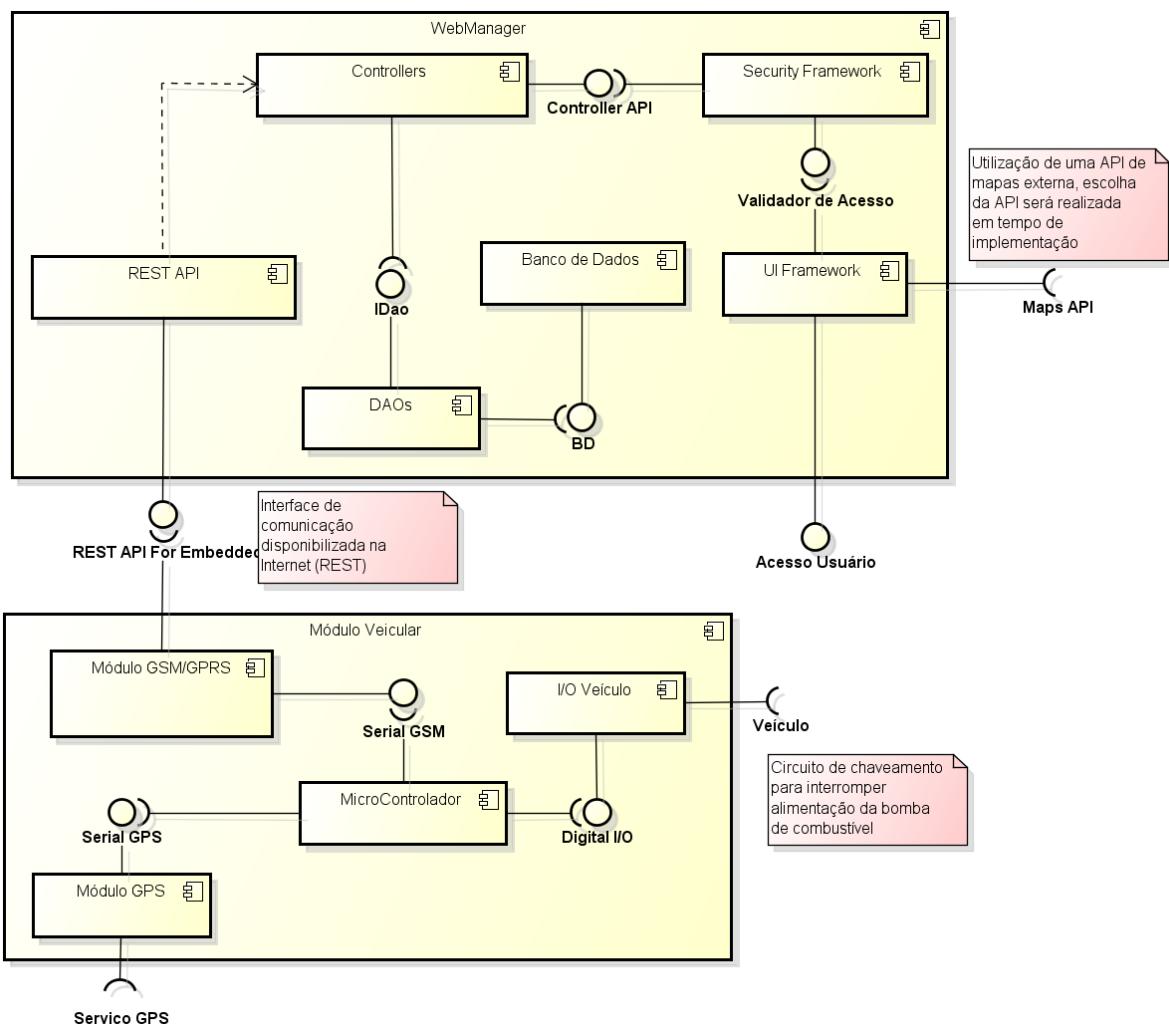


Figura 3.4: Diagrama de Arquitetura do Sistema de Segurança Veicular com uso de GPS baseado em Arduino

Pela definição arquitetural obteve-se um cenário onde existem dois componentes macro: o módulo veicular e o módulo Web chamado WebManager.

O módulo veicular pode ser dividido em diversos subcomponentes, interfaces requeridas e interfaces fornecidas conforme a descrições a seguir.

- Serviço GPS: representa a utilização do serviço de rádio fornecido pelos satélites/estações GPS;
- Módulo GPS: hardware que implementa as funcionalidade GPS, se comunicando com o microcontrolador por um canal serial descrito como Serial GPS;
- Microcontrolador: componente físico que ATmega328 da placa Arduino;
- Digital I/O: Pinos digitais do microcontrolador que interagem com o I/O Veículo;
- I/O Veículo: Hardware que implementa o chaveamento para desligamento do veículo;
- Módulo GSM/GPRS: hardware que implementa funcionalidade de celular para acesso a rede GPRS. Se comunica com o microcontrolador por meio de um canal serial nomeado Serial GSM;

O módulo nomeado WebManager realiza o gerenciamento e aplicação das regras de negócio, interface com usuário e disponibilização do webservice. Seus principais componentes são:

- REST API For Embedded: interface pela qual o módulo embarcado se comunica com o WebManager, é um webservice RESTful pronto para receber e responder às requisições dos módulos;
- Controller: componente que representa o conjunto de controllers (MVC) que atuam dentro do WebManager, manipulando requisições de módulos e usuários;
- IDao: Interfaces para acesso aos DAOs, os controllers possuem instâncias dessas interfaces para obter acesso ao banco;
- DAOs: Implementação dos DAOs que contém os scripts de consulta ao banco de dados;

- Banco de Dados: representação do conjunto de dados relacionais do WebManager armazenados;
- Controller API: Interface para os controllers por onde estes oferecem serviços a outros componentes;
- Security Framework: Componente responsável pela autenticação, autorização e controle de acesso ao sistema web. Utilizado para controle de perfis de usuários;
- UI Framework: Framework utilizado para implementação das views do sistema (páginas HTML);
- Maps API: Indica necessidade de acesso a um serviço de mapas de terceiros para representação visual no sistema.

3.5 Hardware do módulo embarcado

Conforme abordado no capítulo 2, a plataforma de prototipagem Arduino foi escolhida para servir de base para implementação em hardware do módulo embarcado, sendo que a placa controladora Arduino Uno é o núcleo funcional do mesmo.

Baseado no modelo de classes da figura 3.3, é possível perceber que existe uma correspondência de um para um entre classe e hardware que implementa certa funcionalidade, assim sendo:

- Classe ModuloVeicularHw: corresponde a placa controladora Arduino Uno.
- Classe GSM: corresponde em hardware ao Shield GSM que está acoplado a placa controladora.
- Classe GPS: corresponde ao Shield GPS com módulo GPS EM406 descrito no capítulo 2.
- IO Veículo: correspondente ao módulo de relé para acionamento de cargas em conjunto com um pino digital da placa controladora para realizar leitura do estado do alarme do veículo.

3.5.1 Placa controladora

O processo de montagem se inicia com a placa microcontroladora que recebe os shields conforme o desenvolvimento do firmware. A figura 3.5 mostra o exemplar de Arduino Uno adquirido para montagem do protótipo.



Figura 3.5: Exemplar Arduino Uno

Por serem parte de uma plataforma orientada a prototipagem, os hardwares da família Arduino possuem algumas características para facilitar a integração entre módulos, placas de circuito impresso e componentes de outra natureza, é possível enumerar as principais:

1. Layout padrão: os shields seguem o mesmo desenho da placa controladora Arduino Uno, com isso não existe desproporcionalidade de tamanho na integração.
2. Pinos para empilhamento: as placas possuem pinos passantes que permitem conectar nós de circuito a uma segunda placa empilhada sob a primeira. Os pinos são mostrados na figura 3.6.
3. O kit Arduino traz um conjunto de jumpers com terminais do tipo pino, facilmente conectáveis aos soquetes da placa controladora.
4. Textos explicativos impressos sobre as placas descrevem a função de cada pino, diminuindo riscos de conexões erradas que podem levar à queima de componentes.

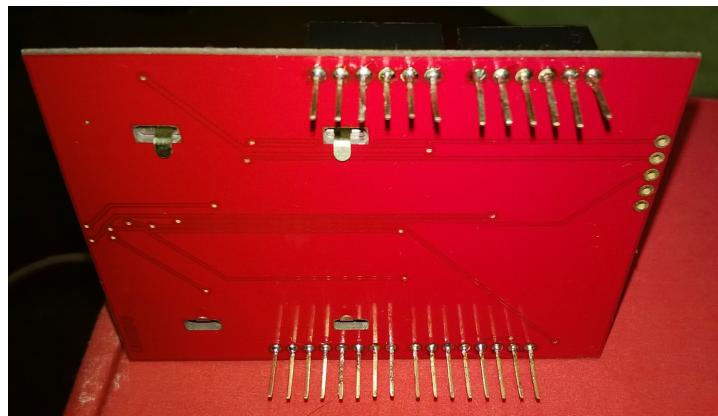


Figura 3.6: Pinos para empilhamento no Shield GSM

3.5.2 Shield GSM

O próximo componente integrado é o Shield GSM, nesta etapa apenas é necessária a soldagem dos pinos de empilhamento conforme a figura 3.6, e encaixar o conjunto sobre a placa controladora.

Conforme obtido na documentação disponibilizada pelo fabricante, este shield utiliza alimentação dos pinos 5V e 3.3V, além disso faz uso dos pinos digitais 2 e 3 para abertura de um canal serial com o microcontrolador, o restante dos pinos fica livre para uso. Existe um conector para antena padrão SMA, que recebe uma antena quad band afim de melhorar a recepção de sinal da rede GSM. A figura 3.7 mostra a integração realizada.



Figura 3.7: Shield GSM integrado ao Arduino Uno

3.5.3 Shield GPS

A próxima etapa é a integração do Shield GPS no conjunto atual, para isto se procede a soldagem dos pinos de empilhamento da mesma forma que no passo anterior, porém foi percebido na análise documental que existe um pequeno empecilho que demanda uma adaptação no hardware.

A questão é que o Shield GPS, assim como o GSM, também necessita de um canal serial com o microcontrolador. Em sua configuração padrão, este componente possui uma chave que permite alternar o canal serial entre os pares de pino 0-1 e 2-3, neste momento surge o problema:

- Os pinos 0 e 1 formam o canal serial utilizado para programação e debug do microcontrolador via porta USB, portanto não é possível utilizá-lo nesta configuração.
- Os pinos 2 e 3 são utilizados pelo Shield GSM e não se pode compartilhar o mesmo canal.

A solução aplicada foi alterar os pinos do canal serial para o par 4-5 que se encontra livre. O desenho do shield permite fazer isto facilmente: conforme a figura 3.8, existem dois pontos de solda no circuito com os textos "2" e "3", deve-se remover o pontos de solda desses locais e realizar um jumpeamento do ponto 2 para o pino 4 e do ponto 3 para o pino 5. A chave de seleção deve ser mantida na posição DLINE.

Após a modificação se procede o empilhamento da placa normalmente, deixando o conjunto com o aspecto da figura 3.9.

3.5.4 Circuito de I/O com Veículo

Um dos requisitos do sistema é a possibilidade de bloquear, ou seja, desligar o veículo e não permitir que o mesmo seja ligado novamente por meios normais. Após pesquisar sobre os métodos utilizados pelos alarmes automotivos para realizar este desligamento foi constatado que os mesmos comumente realizam uma intervenção sobre o circuito da bomba de combustível do veículo.

O funcionamento desta técnica se baseia no princípio de malhas fechadas, a figura 3.10 representa o circuito de uma bomba de combustível automotiva: uma vez que a ignição fecha o circuito, a bomba fica alimentada e funciona normalmente.

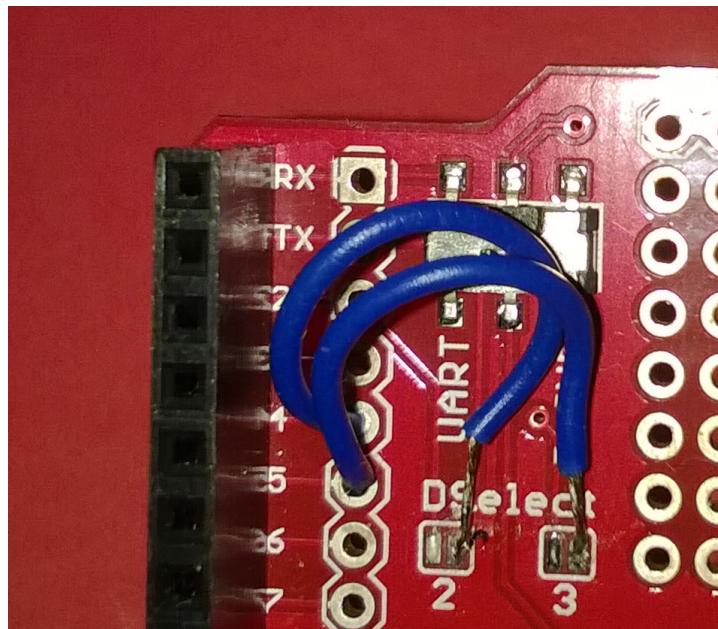


Figura 3.8: Modificação no Shield GPS



Figura 3.9: Integração Arduino Uno, GSM e GPS

A intervenção se dá na inserção de uma nova chave, neste caso um relé normalmente fechado¹ é colocado em série com a chave de ignição, no momento que o microcontrolador acionar o relé, o circuito é aberto e a bomba deixa de funcionar por falta de alimentação.

¹Um relé normalmente fechado permite a passagem de corrente entre dois pontos e, quando acionado, desconecta estes dois ponto impedindo a condução e abrindo o circuito.

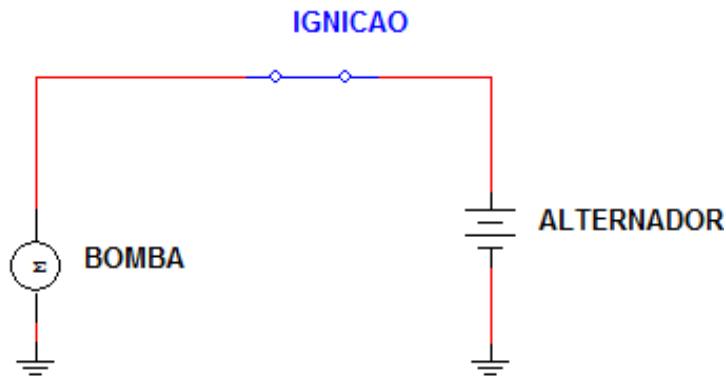


Figura 3.10: Circuito normal da bomba de combustível

A figura 3.11 mostra o circuito alterado e a chave representativa do relé se encontra aberta, indicando um bloqueio do veículo.

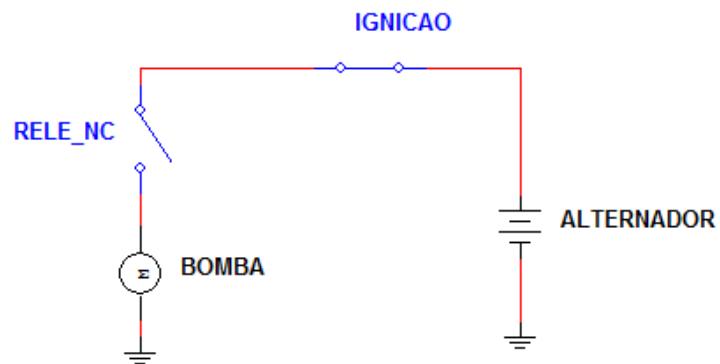


Figura 3.11: Circuito da bomba após intervenção

A bomba de combustível de um automóvel possui uma corrente de operação de 5 a 6 ampéres, com pico de 10 ampéres, conforme relatado no capítulo 2, um microcontrolador necessita de um circuito externo, em geral feito com relés, para manipular esta carga. Para suprir esta necessidade foi adquirido um circuito com relé da fabricante Sparkfun. Detalhes deste componente se encontram no capítulo 2.

A integração deste circuito ao conjunto montado até o momento é feita por meio de três jumpers:

- Jumper ligado em 5 V para alimentação da bobina do relé.
- Jumper ligado em GND (terra) também para a bobina; Jumper de controle ligado no pino digital 12 para sinalizar o acionamento ou desligamento do relé.

Uma intervenção no circuito de relé foi necessária pois por padrão o conector da placa está interligado à malha normalmente aberta, com isso foram soldados dois cabos diretamente à malha normalmente fechada conforme a figura 3.12.



Figura 3.12: Fios interligados à malha normalmente fechada do relé

Ao final jumper também foi adicionado interligando o pino digital 9 ao pino RST para fins de reboot automático do microcontrolador quando necessário. A figura 3.13 mostra o conjunto completo montado.

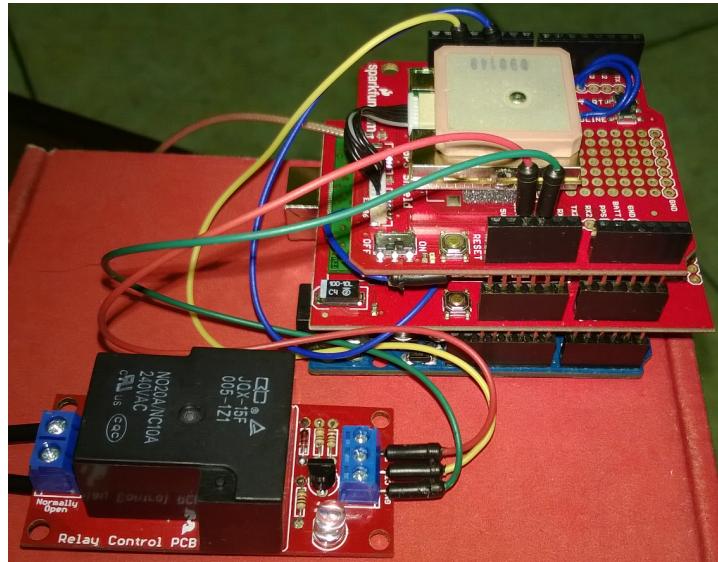


Figura 3.13: Conjunto de hardware do módulo embarcado montado

3.6 Desenvolvimento do Firmware

O módulo embarcado possui um software de propósito específico, ou firmware, gravado na memória flash do microcontrolador, cujas principais funções são:

- Comunicação e obtenção de dados GPS;
- Montagem do corpo da requisição ao webservice;
- Controle da conexão e transmissão de dados GPRS;
- Manipulação da resposta e tratamento de bloqueio/desbloqueio do veículo através do módulo de IO;

A criação desta rotina segue um modelo de atividades bem definidas, que se restringem a um laço de repetição principal. O modelo de programação da plataforma Arduino engloba a implementação de duas funções principais:

- void setup(): função que é executada apenas uma vez a cada inicialização ou reboot da placa controladora, utilizada para inicialização de variáveis, configuração do modo dos pinos e bibliotecas.
- void loop(): bloco executado após o setup(), fazendo o que o próprio nome diz, um laço de repetição que executa enquanto o microcontrolador estiver ligado, permitindo a leitura e modificação de estado. Utilizado para realizar o controle efetivo da placa controladora.

O passos fundamentais do firmware no módulo embarcado são descritas no diagrama de atividades da figura 3.14, o fluxo principal se resume, após a execução da função setup (onde é aberta a conexão com o webservice), na repetição da obtenção dos dados GPS, envio dos dados ao servidor e leitura da resposta, verificando se deve se efetuar o bloqueio do veículo ou não.

Para realização das atividades que envolvem o uso dos shields e outros hardwares acoplados a placa controladora, é necessário criar uma interface de comunicação com os mesmos, que é feita na maioria dos casos, utilizando uma biblioteca fornecida pelo fabricante ou pela comunidade Arduino. Estas bibliotecas são conjuntos de rotinas

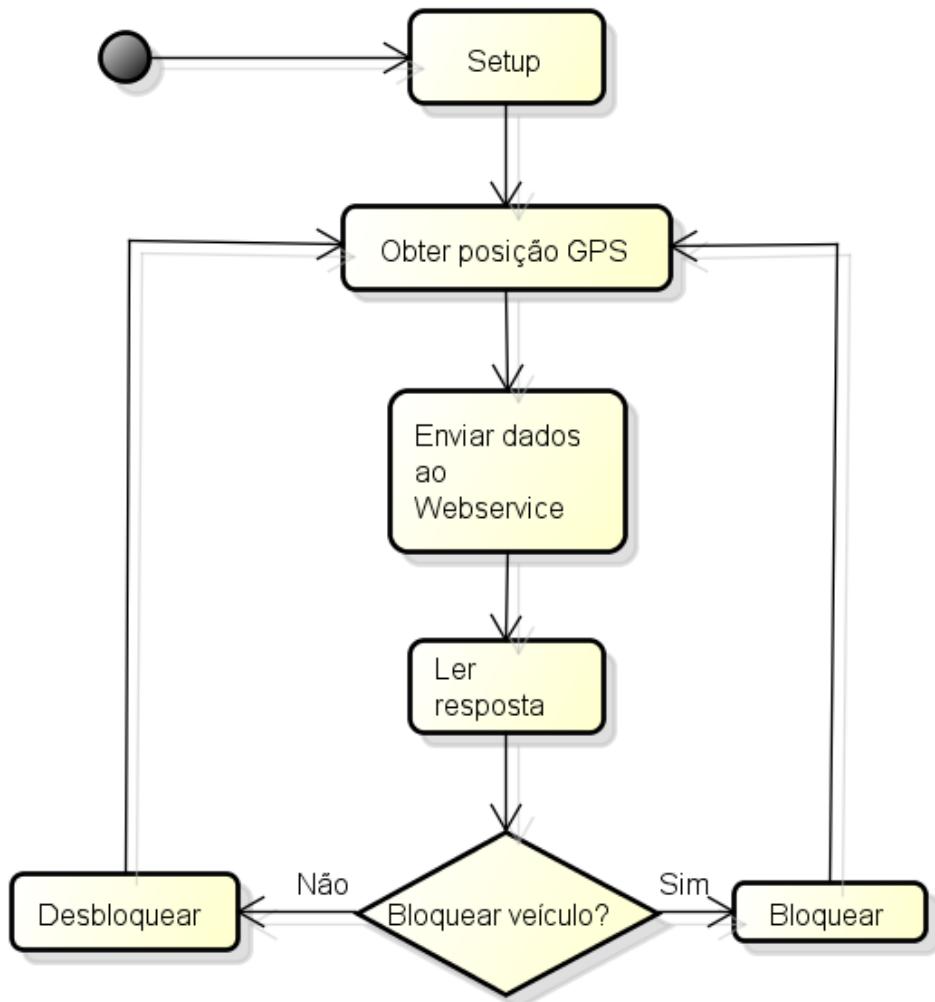


Figura 3.14: Sequência lógica do firmware

escritas em C/C++ e integradas ao programa principal por meio de uma diretiva `#include<biblioteca.h>`.

As subseções a seguir descrevem a realização da interface com os shields GPS e GSM, além da comunicação com o módulo de IO (relé) e uso da funcionalidade EEPROM, que permite persistir um dado em memória não-volátil, recuperando seu estado em caso de reinicialização da placa.

3.6.1 Interface com EEPROM

O microcontrolador da placa Arduino possui uma memória do tipo EEPROM: Electrically-Erasable Programmable Read-Only Memory, com tamanho de 1024 bytes no ATmega328. Neste tipo de memória os valores armazenados são mantidos mesmo que a placa seja desligada, atuando como um disco rígido para armazenamento não-volátil.

Este componente é essencial para realizar a persistência do estado de bloqueio do módulo embarcado, evitando que por um simples desligamento e religamento da bateria ocorra o desbloqueio do automóvel. A cada boot o último estado é restaurado e a cada alteração do mesmo, há uma atualização do valor na EEPROM.

A manipulação da EEPROM consiste na leitura e na escrita de dados, para ambos é necessário incluir o cabeçalho `#include<EEPROM.h>`.

A escrita de dados é feita pela função `EEPROM.write(endereço, valor)`, onde endereço é um inteiro de 0 a 1023 e valor é um byte, aceitando valores de 0 a 255. Para leitura, a função `EEPROM.read(endereço)` é usada, bastando passar o valor do endereço como parâmetro e obter um byte como resposta.

O processo de leitura do último estado e de escrita do estado atual do módulo na memória EEPROM é mostrado no código 3.6.1.

```

1 #include <EEPROM.h>
2
3 #define STATE_PERM_DATA_ADDR 0
4 #define STATE_UNBLOCKED LOW
5 #define STATE_BLOCKED HIGH
6
7 byte moduleState = STATE_UNBLOCKED;
8
9 void setup()
10 {
11     //... Demais configuracoes
12     moduleState = EEPROM.read(STATE_PERM_DATA_ADDR);
13 }
14
15 void treatServerResponse(String *response)
16 {
17     if (response->substring(17, 20)==BLOCK_MESSAGE)
18     {
19         if(moduleState!=STATE_BLOCKED)
20         {
21             moduleState=STATE_BLOCKED;
22             EEPROM.write(STATE_PERM_DATA_ADDR, moduleState);
23             //Comando para bloqueio aqui
24         }
25     }
26     if(response->substring(17, 20)==UNBLOCK_MESSAGE)
27     {
28         if(moduleState!=STATE_UNBLOCKED)
29         {
30             moduleState=STATE_UNBLOCKED;
31             EEPROM.write(STATE_PERM_DATA_ADDR, STATE_UNBLOCKED);
32             //Comando para desbloqueio aqui
33         }
34     }
35 }
36 //... Demais definicoes, inclusive void loop()

```

Código 3.6.1: Rotinas de leitura e escrita em EEPROM

3.6.2 Interface com I/O

A interface com o módulo de I/O com relé é bastante simplificada, uma vez que sua integração é realizada por meio dos pinos digitais. O processo basicamente define que o pino de comunicação utilizado tenha seu modo de operação (entrada ou saída) definido na função setup, depois disso, basta realizar uma chamada da função digitalWrite(pino, estado), onde o parâmetro pino é o número do mesmo assim como descrito na impressão sobre o circuito impresso, e o parâmetro estado pode assumir o valor da macro HIGH (ligado) ou LOW (desligado). O código 3.6.2 mostra a configuração e uso do pino de digital para acionamento ou desacionamento do relé de controle de carga.

```

1 #define IO_PIN 12
2 #define STATE_UNBLOCKED LOW
3 #define STATE_BLOCKED HIGH
4
5 void setup()
6 {
7     pinMode(IO_PIN, OUTPUT); //Define o modo de operacao do pino
8     //... Demais configuracoes e obtencao do valor de moduleState
9     digitalWrite(IO_PIN, moduleState);
10 }
11
12 void treatServerResponse(String *response)
13 {
14     //..Leitura da resposta
15     if(moduleState!=STATE_BLOCKED)
16     {
17         moduleState=STATE_BLOCKED; //HIGH
18         //...Demais rotinas
19         digitalWrite(IO_PIN, moduleState);
20     }
21     if(moduleState!=STATE_UNBLOCKED)
22     {
23         moduleState=STATE_UNBLOCKED; //LOW
24         //...Demais rotinas
25         digitalWrite(IO_PIN, moduleState);
26     }
27 }
28 //... Demais definicoes, inclusive void loop()

```

Código 3.6.2: *Rotinas de acionamento do pino digital para controle do relé*

3.6.3 Interface com Shield GPS

O Shield GPS fornece está acoplado fisicamente à placa Arduino Uno e se comunicando por um canal serial estabelecido nos pinos 4 e 5 conforme abordado na seção 3.5.3, entretanto é preciso configurar este canal na inicialização do controlador e obter os dados enviados pelo módulo.

Para obtenção dos dados de posicionamento, dois artefatos necessitam estar presentes no código:

- Um canal serial lógico: feito utilizando a biblioteca SoftwareSerial, utilizado para efetuar a troca de dados bidirecional entre o microcontrolador e o módulo GPS.
- Um parser das mensagens GPRMC: a biblioteca TinyGPS disponibilizada pela comunidade realiza esta função, capturando as mensagens transmitidas pelo módulo e disponibilizando as informações

Além de configurar os itens citados, é necessário que o módulo tenha completado a montagem da mensagem GPRMC (citada no capítulo 2), para isto é criada uma função auxiliar chamada feedGps() que faz uma checagem contínua no status da mensagem por

meio de um laço de repetição, que é interrompido caso a mensagem esteja completa ou ocorra um timeout sem resposta.

O Código 3.6.3 mostra as chamadas necessárias para capturar a posição atual do módulo GPS.

```
1 #include <TinyGPS.h>
2 #include <SoftwareSerial.h>
3
4 #define GPS_T_OUT 5000
5 #define GPS_TX_PIN 4
6 #define GPS_RX_PIN 5
7
8 TinyGPS gps;
9 SoftwareSerial gpsCommunicator(GPS_TX_PIN, GPS_RX_PIN);
10 float actualLatitude = 0.0f;
11 float actualLongitude = 0.0f;
12
13 static bool feedGps()
14 {
15     unsigned long checker = millis();
16     while (true)
17     {
18         if (gpsCommunicator.available() && gps.encode(gpsCommunicator.read()))
19             return true;
20         if((millis()-checker)>GPS_T_OUT)
21             return false;
22     }
23 }
24
25 void setup()
26 {
27     //... Demais configurações
28     gpsCommunicator.begin(4800); //4800 bps de baudrate
29 }
30
31 void loop()
32 {
33     gpsCommunicator.listen();
34     if(feedGps())
35         gps.f_get_position(&actualLatitude, &actualLongitude);
36     //... Demais rotinas
37 }
```

Código 3.6.3: Rotinas de obter a posição geográfica via GPS

3.6.4 Interface com Shield GSM

O módulo SM5100B do Shield GSM 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 porém 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.

No SDK Arduino existe uma biblioteca chamada SoftwareSerial com uma classe principal de mesmo nome, a qual abstrai os comportamentos de um elemento que se comunica com o microcontrolador por um canal serial, facilitando questões como a configuração de pinos RX e TX, transmissão e recepção serializada de dados, controle de buffer e outras questões pertinentes.

Foi implementada uma classe de comunicação com o módulo utilizando o conceito de herança do paradigma de orientação a objetos, onde a classe Shield GSM é uma subclasse de SoftwareSerial.

A figura 3.15 mostra o diagrama da classe de comunicação e a herança de SoftwareSerial.

3.6.5 Requisição ao webservice

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.

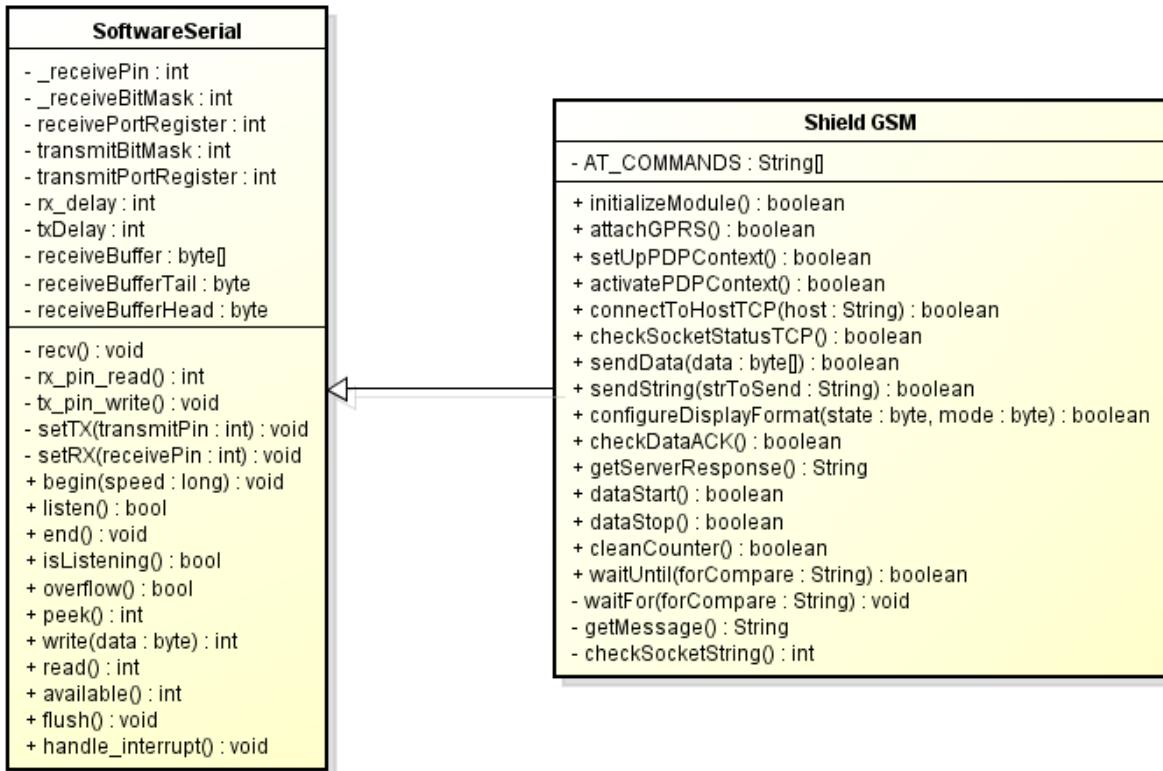


Figura 3.15: Classe de comunicação com SM5100B

A figura 3.16 mostra a mensagem enviada pelo módulo veicular a um servidor de testes, é possível observar o cabeçalho HTTP simulando um browser com engine Mozilla/5.0 e a sinalização do uso de dados estruturados JSON.

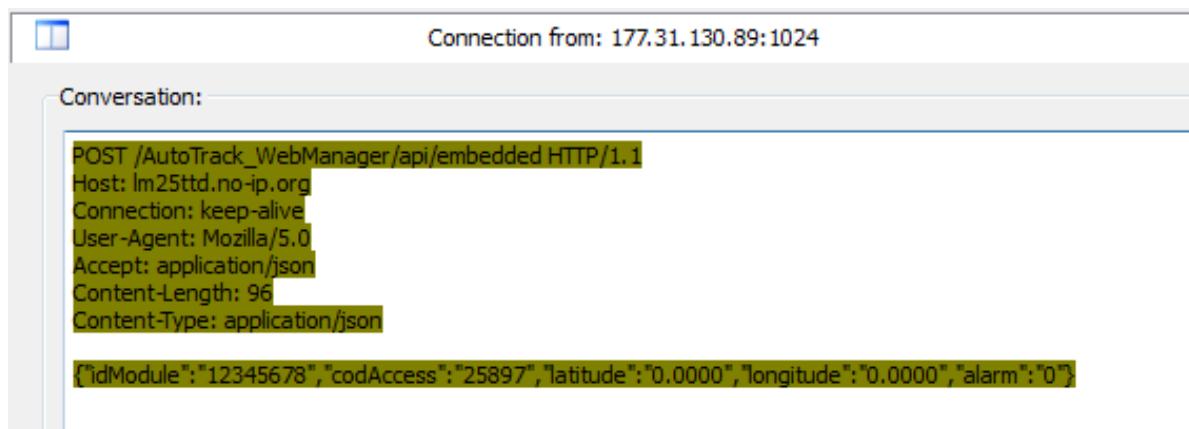


Figura 3.16: Requisição HTTP feita pelo módulo veicular

3.6.6 Visão geral do funcionamento do módulo embarcado

Conforme descrito anteriormente, o firmware opera em duas funções principais, a função void setup() e a função void loop(). A figura 3.17 mostra as atividades realizadas durante o boot, especificamente passos que envolvem a configuração para acesso a rede de dados GSM, login no APN da operadora, permitindo assim que seja realizada posteriormente a conexão e troca de dados com o webservice.

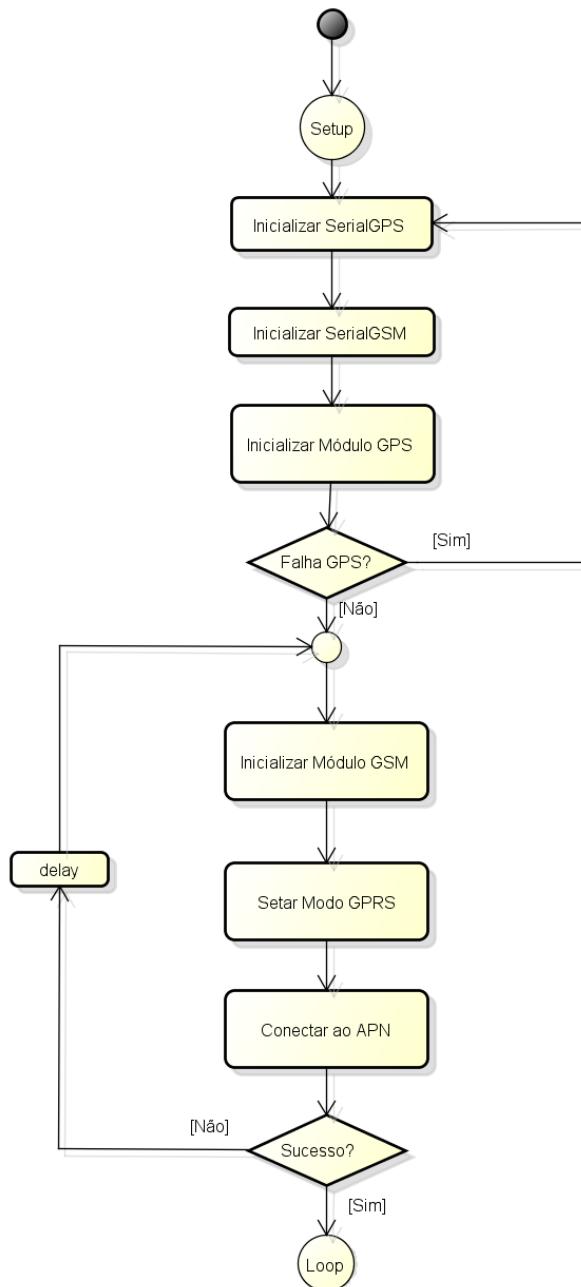


Figura 3.17: Processo de setup

Na figura 3.18 as atividades durante o loop envolvem a obtenção dos dados GPS, montagem da requisição, conexão e envio de dados ao webservice, em caso de falha há um direcionamento para a função de setup, onde a configuração inicial é aplicada, retomando o funcionamento normal.

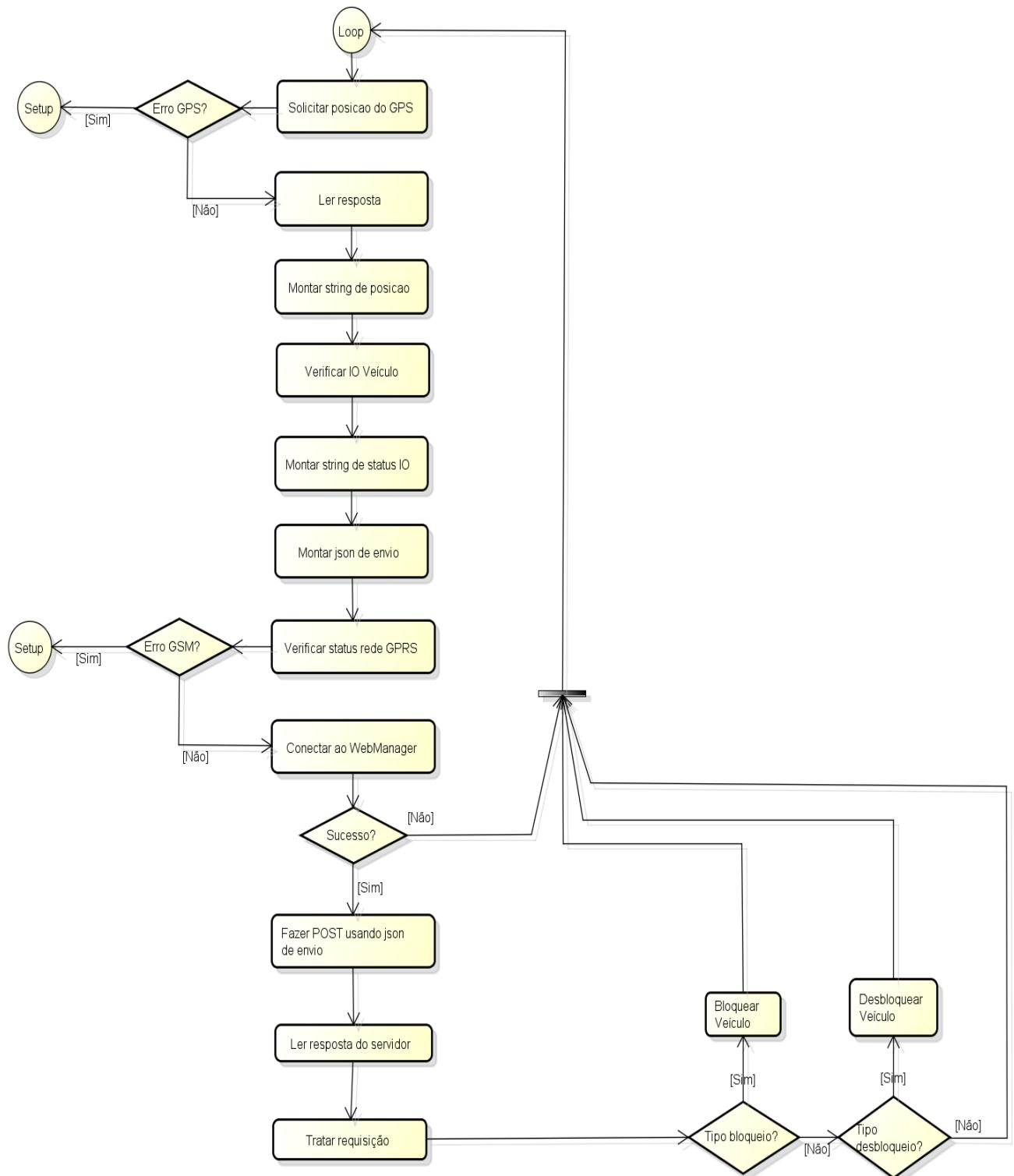


Figura 3.18: Atividades durante o loop

3.7 Módulo Web (WebManager)

O módulo web, definido também como WebManager, é a aplicação que roda na internet responsável por receber e armazenar 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.

A técnica REST permite fazer mapeamentos de métodos diretamente em URL com correspondência aos métodos HTTP, um 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 segurança contra falhas, por exemplo no desligamento repentino do sistema.

3.7.1 Modelo geral

A arquitetura da modelagem do WebManager foi baseada no padrão MVC que segundo Mendes [15] 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 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), DAO, Factory Method descritos por Gamma [8], foram aplicados pois o uso de padrões de projeto fornece soluções arquiteturais otimizadas

para problemas conhecidos na engenharia de software. A Figura A.2 nos anexos mostra o diagrama de classes obtido.

3.7.2 Camada de modelo

A camada de modelo representa os dados da aplicação e as regras do negócio que governam o acesso e a modificação dos dados. O modelo mantém o estado persistente do negócio e fornece ao controlador a capacidade de acessar as funcionalidades da aplicação encapsuladas pelo próprio modelo. As classes definidas nesta camada também são chamadas de entidades, o diagrama de classes para o modelo deste trabalho é mostrado na figura 3.19.

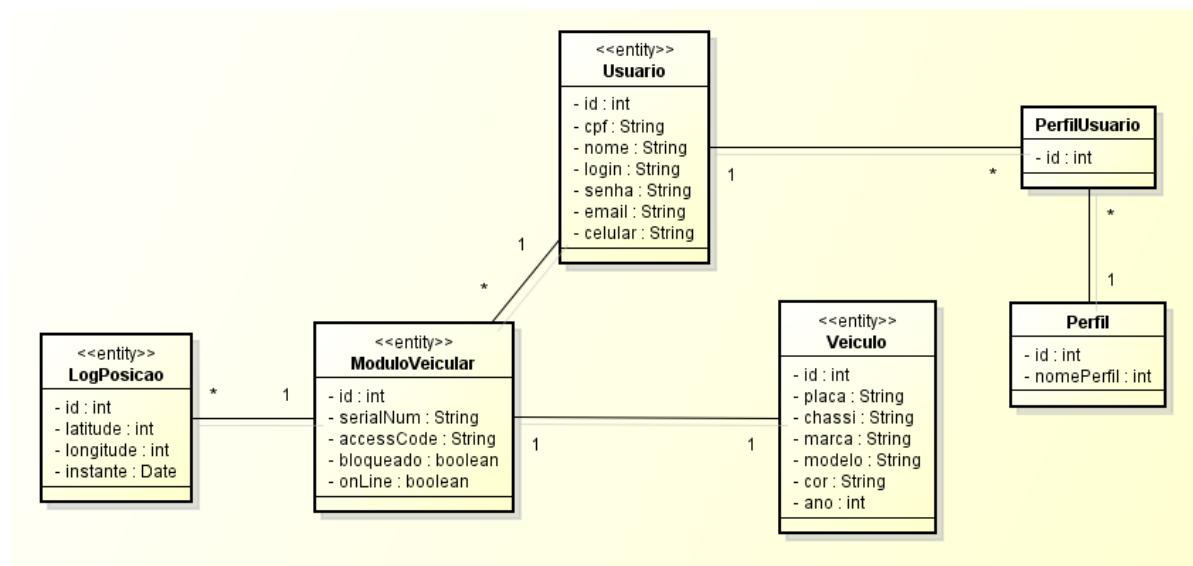


Figura 3.19: Model - Entidades do sistema

Uma vez que o conceito define que esta camada guarda um estado persistido, o diagrama de classes gera naturalmente uma modelagem de banco de dados. A criação de uma base de dados relacional a partir deste modelo é facilitada com o uso de um framework de mapeamento objeto-relacional (object-relational mapping ou simplesmente ORM).

O framework Hibernate ORM foi adotado, adicionado alguns aspectos que facilitam o desenvolvimento de aplicações complexas, os principais são:

- A base de dados é gerada automaticamente a partir das classes de modelo, eliminando o trabalho da criação de scripts DDL para criação de tabelas, chaves

estrangeiras, constraints e outros aspectos inerentes à definição de banco de dados.

- A necessidade da criação de scripts SQL para consulta de dados é eliminada, pois se trabalha diretamente com objetos Java inter-relacionados. Ao se obter um atributo de um objeto, seja ele um atributo simples ou complexo, o framework gera em tempo de execução o script de consulta ao banco, abstraindo uniões, subconsultas e outras necessidades.
- O gerenciamento de transações é feita de maneira automatizada, bastando anotar os métodos transacionais para que haja este controle efetivo.
- Caso seja necessário criar consultas manuais, o Hibernate possui uma linguagem própria bastante intuitiva, chamada HQL, onde se trabalha facilmente com a combinação de atributos de classes para se obter informação desejada.

O diagrama da figura 3.19 foi traduzido em uma base de dados com seis tabelas, são elas:

1. tb_logposicao, que guarda a posição geográfica de cada requisição feita pelos módulos veiculares.
2. tb_moduloveicular, contém todos os módulos veiculares embarcados cadastrados no sistema.
3. tb_perfil, guarda os perfis de usuários do sistema, no caso administrador ou usuário comum.
4. tb_usuario, armazena todos os usuários cadastrados no sistema.
5. tb_perfilusuario, associa usuários e perfis.
6. tb_veiculo, contém todos os veículos cadastrados no sistema.

Compondo a camada de modelo existe uma implementação que aplica o padrão DAO(Data Access Object) descrito no capítulo 2. Neste trabalho a camada DAO implementa algumas classes que contém métodos definidos no framework Hibernate para operações CRUD, conforme a figura 3.20.

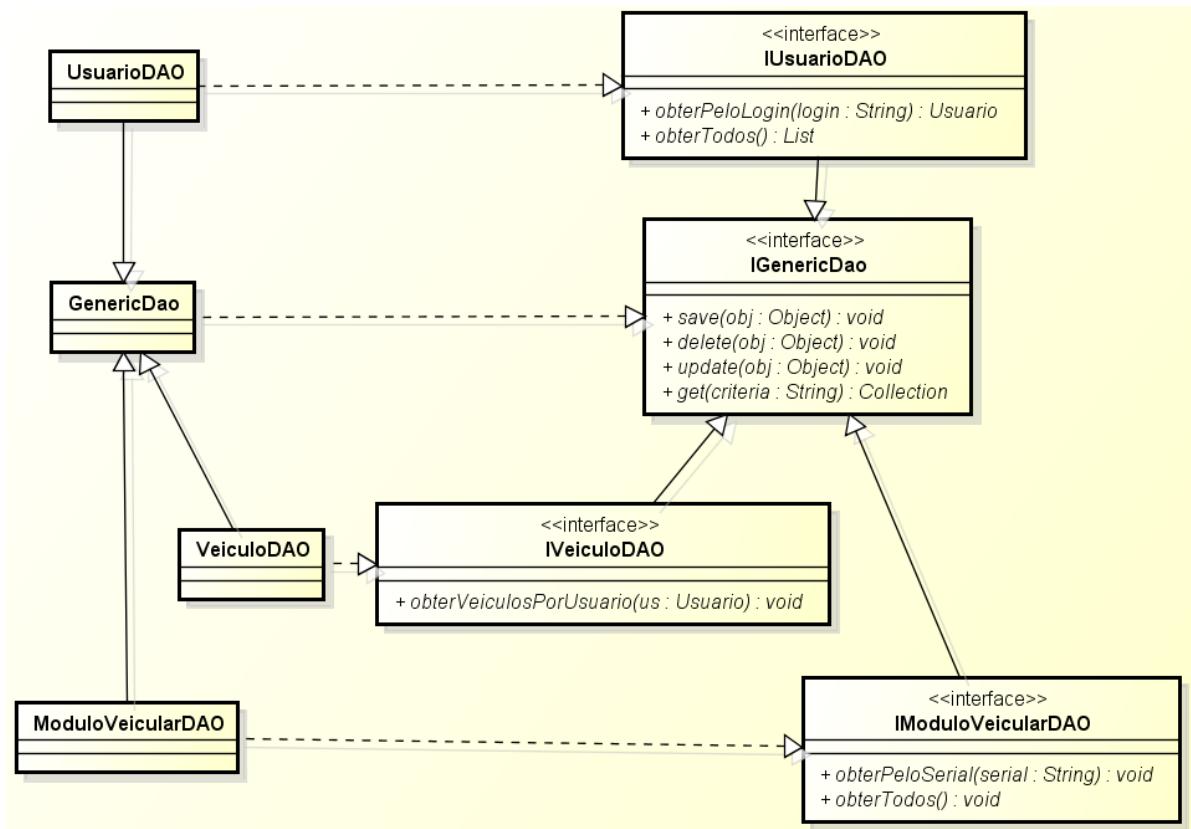


Figura 3.20: Classes DAO

O par de interface/classe respectivamente IGenericDAO/GenericDao, contém métodos para operações CRUD anotados com as respectivas configurações de transação. Os principais métodos da implementação genérica são:

- `save(object)`: Salva um objeto de uma classe definida como entidade no banco de dados.
- `delete(object)`: Remove do banco de dados o objeto passado como parâmetro.
- `update(object)`: Atualiza os atributos de um objeto previamente salvo no banco de dados.
- `getAll(class)`: Retorna uma lista contendo todos os objetos do tipo da classe passada como parâmetro que estejam salvos no banco de dados.

As classes especializadas **VeiculoDAO**, **ModuloVeicularDAO** e **UsuarioDAO**, além de suas respectivas interfaces, apenas estendem a implementação genérica com a inclusão

de métodos de busca específicos, utilizados para alguns cenários.

Cada controller contém como atributo uma interface para o DAOs correspondente a sua natureza, que sofre injeção de uma instância da classe correlata em tempo de execução. O uso de interface permite que, durante um teste unitário, a implementação real seja substituída por um mock, isolado a classe a ser testada de dependências externas.

3.7.3 Camada de controle

Nesta camada são definidos os controladores (controllers) do sistema, um controlador define o comportamento da aplicação, é ele que interpreta as ações do usuário e as mapeia para chamadas do modelo. Em um cliente de aplicações Web essas ações do usuário poderiam ser cliques de botões ou seleções de menus. As ações realizadas pelo modelo incluem ativar processos de negócio ou alterar o estado do modelo.

Na figura 3.21 pode se observar que existe um controller para cada entidade principal do sistema, além de alguns métodos base para alteração do estado persistido do modelo.

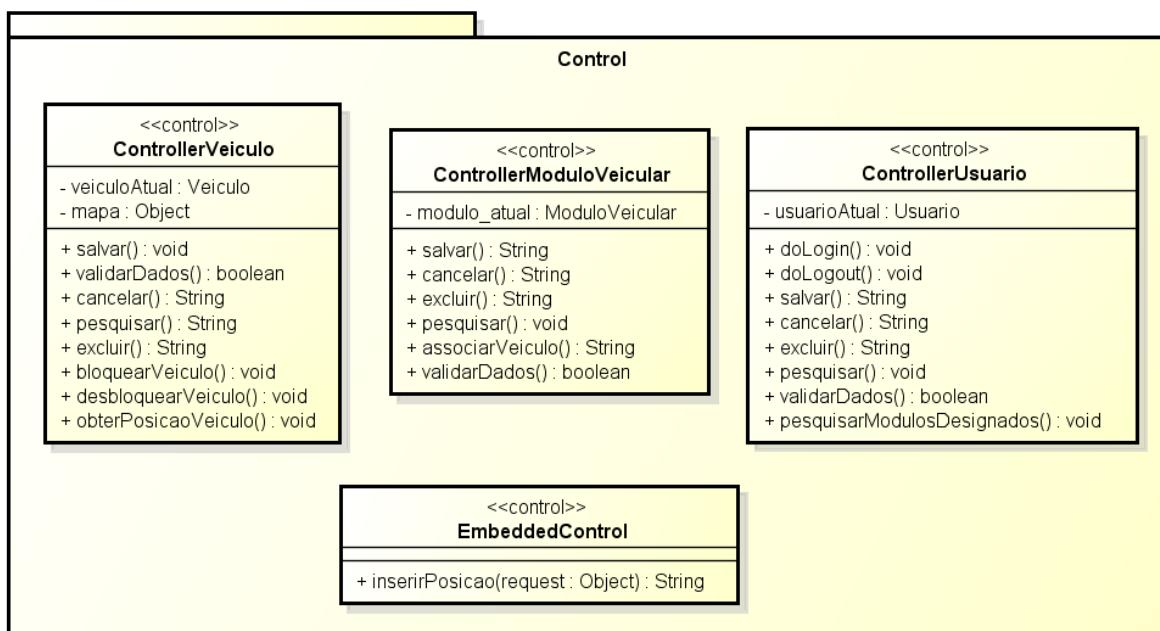


Figura 3.21: Controllers

Os controladores são beans, componentes Java injetados na interface com o usuário permitindo acesso a serviços não apenas da camada de modelo, mas também serviços externos como no caso do Framework de mapas. Neste caso, o controlador ControllerVeiculo guarda uma instância de mapa que é atualizada conforme as informações do módulo

veicular correspondente vão chegando, além disso o controlador atualiza a página web correspondente.

Nesta camada também é feita a integração com o Spring Security, framework que gerencia o Login, Logout e acesso aos serviços de acordo com o perfil do usuário, não permitindo que alguém logado com certo privilégio acesse serviço que não correspondem à sua credencial.

3.7.4 Webservice

O webservice disponibilizado pela aplicação web é uma API RESTful, onde o serviço principal fica disponibilizado na URL `{servidor.url}/AutoTrack_WebManager/api/embedded`.

A figura 3.22 demonstra um exemplo de requisição e mapeamento feito pelo webservice para entidades manipuláveis na camada de controllers do sistema.

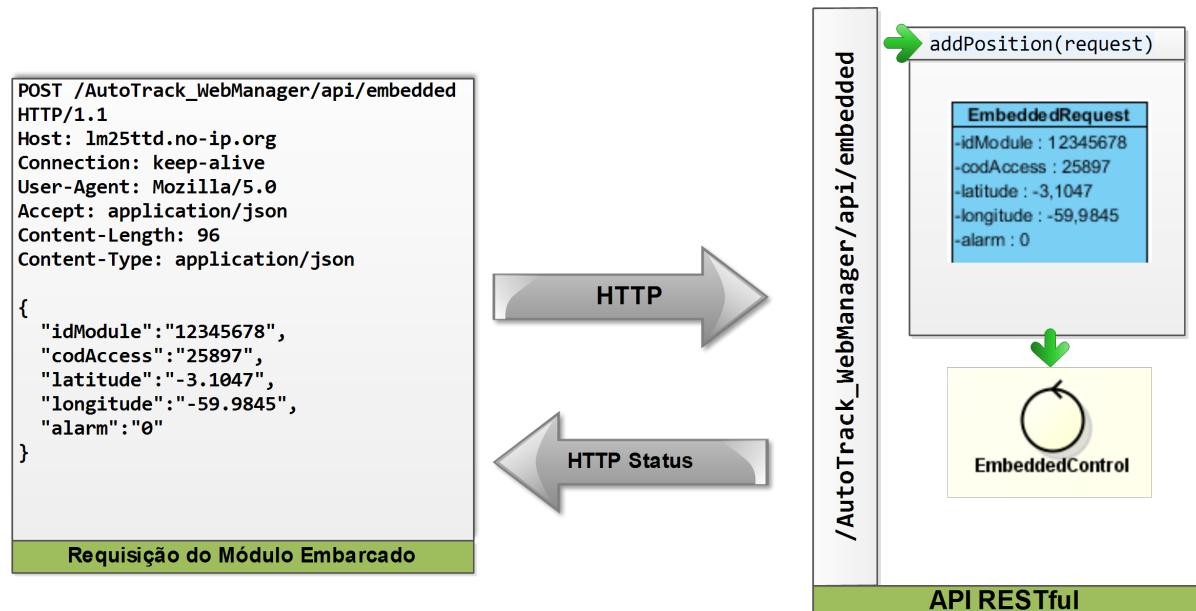


Figura 3.22: Esquema do webservice

O funcionamento do webservice pode ser sintetizado em alguns passos onde existe a interação entre o módulo veicular e a API REST, em concordância com a figura anterior as etapas são as seguintes:

1. O módulo embarcado monta uma requisição HTTP POST para a URL mostrada no esquema, esta requisição traz alguns parâmetros de configuração, como:

- Host: endereço de hospedagem da aplicação;
 - Keep-alive: sinaliza que deverá manter a conexão entre o módulo embarcado e o webservice sempre aberta.
 - User-Agent: indica a engine do browser web, neste caso simula que a requisição está sendo feita de um navegador com engine Mozilla, como o Firefox.
 - Accept e Content-Type: indica o tipo de conteúdo enviado e aceito como resposta, neste caso dados JSON.
 - Content-Length: tamanho em bytes do campo payload.
 - Payload: todo conteúdo compreendido entre as chaves, inclusive as mesmas, se trata de uma estrutura JSON trazendo informações sobre o módulo veicular embarcado utilizadas para alimentar o sistema.
2. A requisição trafega sobre o protocolo HTTP, direcionado a URL em que o webservice está esperando. Esta URL está mapeada para uma função Java chamada addPosition() que recebe como parâmetro um objeto do tipo EmbeddedRequest. Este parâmetro é montado automaticamente pelo framework Spring a partir de um processo de parser da requisição, gerando um objeto EmbeddedRequest.
 3. Uma vez montado o objeto do tipo EmbeddedRequest, este é passado para a classe EmbeddedControl, responsável por montar recuperar do banco de dados as informações referentes ao módulo veicular que está enviando a requisição em questão e, de posse disto, gerar uma nova entrada na tabela LogPosicao. A informação sobre o módulo veicular cruzada com o veículo de instalação mostra se existe ou não um bloqueio do conjunto.
 4. De acordo com o estado de bloqueio, a resposta para o módulo veicular é diferente: caso exista um bloqueio, o webservice responde com status HTTP 200, caso contrário ele responderá com status HTTP 300.
 5. O módulo veicular analisa o status de resposta e age caso necessário.

3.7.5 View

Para implementação da camada de view, que neste escopo são páginas Web, foi utilizado o framework Primefaces, que fornece uma infinidade de componentes de UI

prontos para uso, bastando setar algumas propriedades. As páginas são escritas em XHTML e interpretadas por um container como o Tomcat, que fornece o ambiente de execução para aplicações Java Web.

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. A figura 3.23 mostra a tela de login do sistema.

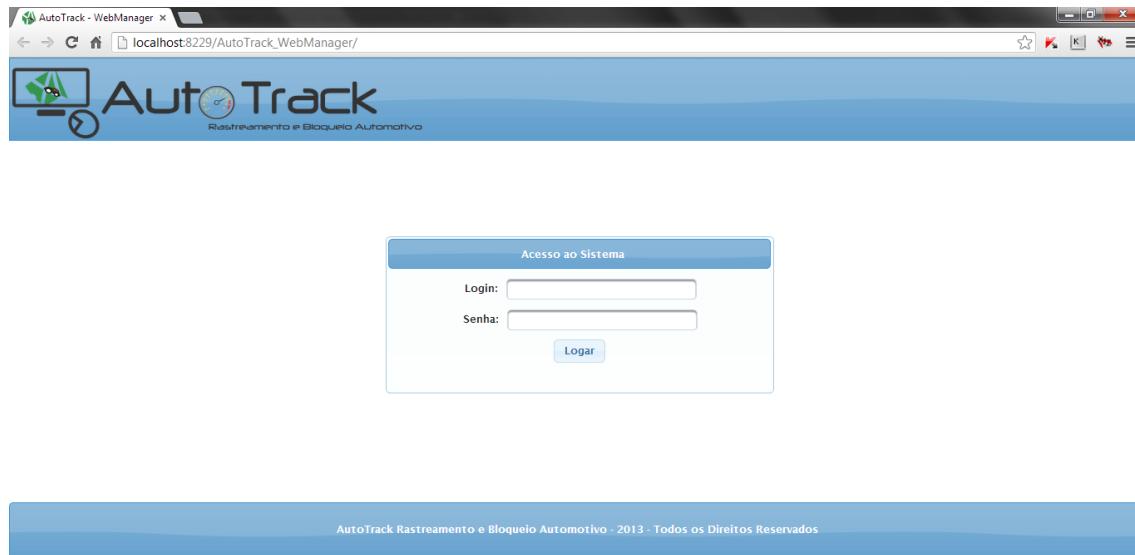


Figura 3.23: Tela de Login do Sistema

O rastreamento do módulo é feito com o auxílio da API do Google Maps. O framework Primefaces possui um componente que realiza esta ligação com a API, trazendo os dados geográficos correspondentes a uma posição latitude-longitude fornecida.

A figura 3.24 exibe a página de rastreamento de veículo, é possível perceber o círculo verde que corresponde a posição aproximada do veículo.

A ação de bloqueio do veículo é realizada por meio do botão "Bloquear", que atualiza o estado do veículo em foco no banco de dados. Com isto, ao realizar a próxima requisição ao webservice, o módulo embarcado no veículo citado receberá uma resposta HTTP correspondente à ação de bloqueio e assim deverá acionar o mecanismo para realizar a ação.

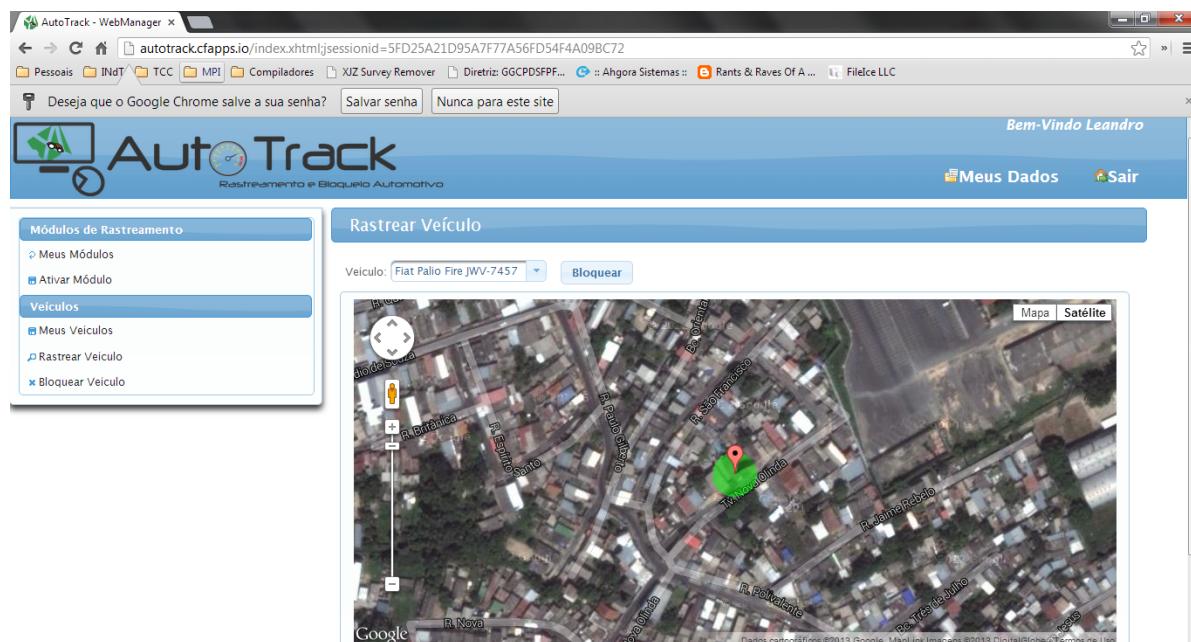


Figura 3.24: Tela de Rastreamento

Capítulo 4

Resultados e Discussões

4.1 Testes de Carga do Módulo Web

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 4.1 mostra o comportamento serrilhado do gráfico de consumo de memória, demonstrando a ausência de leaks e atuação correta do Garbage Collector.

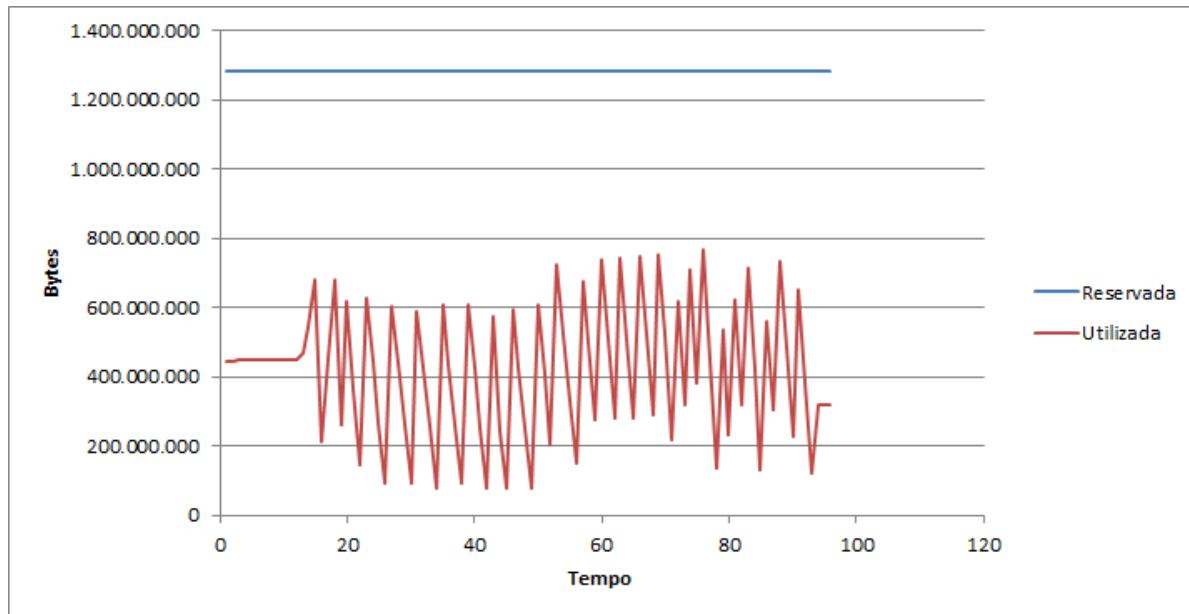


Figura 4.1: Gráfico do consumo de memória heap

O consumo de CPU mesmo com a ação do Garbage Collector ficou em margens coerentes de acordo com o gráfico da figura 4.2.

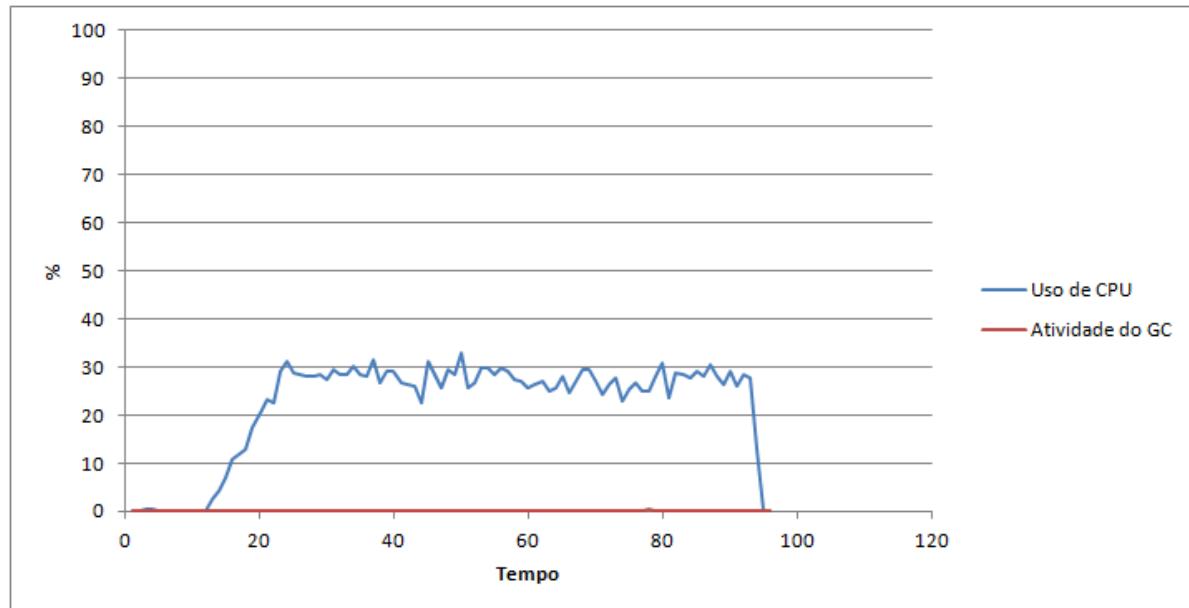


Figura 4.2: Gráfico do uso de CPU

4.2 Integração ao veículo

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 automóvel em movimento, bloqueio e desbloqueio foram realizados com sucesso e registrados em vídeo, anexado em formato digital a este trabalho. A figura 4.3 ilustra 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 4.4 mostra as linhas inseridas no banco de dados durante o período de testes.

| | id [PK] | serie | 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 Resultados

Os resultados gerados por este projeto incluem:

1. Artefatos de modelagem: diagrama casos de uso, diagrama de classes, diagrama de componentes de arquitetura, diagrama de atividades do módulo embarcado e diagramas de sequência.
2. Classe de comunicação com o módulo GSM SM5100B para Arduino.
3. Código fonte do firmware do módulo veicular embarcado, componente do sistema de segurança veicular com uso de GPS baseado em Arduino. Código fonte da aplicação

Web que contempla o *webservice*, lógica de negócio, banco de dados e interface com usuário.

Estes artefatos se encontram em anexo no formato de mídia ótica, possibilitando o desenvolvimento de novos projetos que possuam cenário de operação e requisitos similares.

Ao acessar a mídia, será exibida uma estrutura de diretórios em que os principais artefatos seguem a localização descrita:

- DOC: contém esta monografia em formato digital.
- DESIGN/ModelagemTCC.asta: este arquivo contém toda modelagem do sistema. Deve ser aberta com software Astah ou compatível.
- SRC/Arduino SM5100B Lib/SM5100B_GPRS: contém a classe de comunicação entre o Arduino e o módulo GSM SM5100B.
- SRC/Embedded_GPS_RealTracking: possui o código fonte do firmware desenvolvido para o módulo veicular embarcado. Deve ser editado com o software IDE Arduino ou compatível.
- SRC/AutoTrack_WebManager: contém o código fonte do módulo Web deste projeto. Pode ser importado como projeto na IDE Eclipse.

Capítulo 5

Considerações Finais

5.1 Conclusão

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.

Neste trabalho foi desenvolvido um sistema de segurança veicular com uso de GPS baseado em Arduino, composto por um módulo de hardware embarcado no veículo e um sistema Web, cujo escopo soluciona o problema citado, permitindo ao usuário rastrear e bloquear seu veículo via Internet.

Houve a opção pelo uso da plataforma de hardware livre Arduino, pois conta com diversos componentes desenvolvidos para aplicação imediata, documentação disponibilizada na internet e uma comunidade extensa de desenvolvedores. Ao longo da elaboração do sistema foram gerados artefatos documentais que, juntamente com os códigos-fonte, foram disponibilizados publicamente.

O protótipo foi submetido a testes isolados e testes experimentais, este último contemplou a integração do módulo de hardware a um veículo, permitindo que uma análise de comportamento em ambiente real fosse realizada. O sistema Web foi submetido a testes de performance e exploratórios, obtendo plena aprovação de seu comportamento.

O projeto se mostrou economicamente viável mesmo quando utilizados componentes de prototipagem, portanto uma produção em escala industrial derrubaria o custo e

permitiria a miniaturização do conjunto do módulo veicular embarcado.

De acordo com o estudo e testes realizados, o sistema final conseguiu cumprir com sucesso a proposta e os objetivos que foram apresentados na fase inicial de projeto, tudo isto dentro do tempo disponível e seguindo o cronograma elaborado na mesma proposta.

5.2 Dificuldades encontradas

Ao longo do processo de desenvolvimento alguns entraves foram encontrados, porém alternativas para solução foram aplicadas com sucesso, permitindo a elaboração do sistema completo.

A compra de componentes eletrônicos no mercado local foi um problema devido a ausência de alguns destes para venda, como é o caso de microcontroladores, módulo GPS e módulo GSM. A importação dos produtos foi a solução mais viável, mesmo que o custo financeiro tenha sido elevado.

Conforme citado no capítulo 3, o fabricante do módulo GSM SM5100B não disponibiliza uma biblioteca para comunicação com Arduino, portanto houve a necessidade de se criar tal biblioteca, o que acarretou no investimento de tempo para tal atividade.

A placa Arduino Uno possui uma quantidade de memória RAM bastante limitada, o que gerou a necessidade de diversos ciclos de desenvolvimento e testes (aumentando o tempo de projeto em relação ao estimado inicialmente) até que o firmware fosse otimizado para operar com tal quantidade de memória.

No momento da integração com o veículo, a falta de documentação dos circuitos elétricos do mesmo dificultou a integração do protótipo. Após uma pesquisa em fóruns especializados em alarmes, foi possível localizar o cabeamento a ser alterado para devida integração com o módulo embarcado.

Por fim, a falta de hospedagem gratuita para aplicações Java Web foi um empecilho inicial, superado com criação de um servidor próprio acessível externamente com auxílio da plataforma *no-ip.org*.

5.3 Trabalhos futuros

Existem alguns pontos de melhoria que podem ser aplicados ao sistema atual, otimizando-o e até mesmo gerando um projeto de novo escopo. Como trabalhos fu-

turos sugerem-se algumas propostas:

- Desacoplamento da interface com o usuário do sistema Web deixando somente como serviço de *backend*, permitindo a criação de um cliente mobile que consome dados deste serviço.
- Registrar os pontos geográficos comuns do veículo, criando um túnel virtual. Caso o veículo se desvie deste túnel virtual um alarme será emitido.
- Utilizar uma estratégia de comunicação assíncrona entre o módulo embarcado e o *webservice*, como uso de Websockets na placa Arduino. Esta abordagem pode melhorar o tempo de resposta do sistema.
- Criar um circuito de alimentação elétrica de *backup*, mantendo o módulo embarcado funcionando caso haja desligamento da bateria do automóvel.
- Monitoramento do acionamento do alarme com envio de SMS ou realização de chamada telefônica ao proprietário em caso de anomalias.
- Comunicação direta com celular, possibilitando o bloqueio automático do veículo caso o proprietário leve o aparelho móvel consigo.
- Integração de uma câmera para capturar imagens do interior do veículo e enviá-las a aplicação Web, facilitando a identificação de criminosos.

5.4 Disciplinas aplicadas

O desenvolvimento do projeto demandou aplicação do conhecimento de diversas disciplinas integrantes do curso de Engenharia de Computação, as principais foram:

- Lógica e linguagem de programação
- Circuitos elétricos
- Eletrônica analógica e digital
- Redes de computadores
- Projeto de sistemas embarcados

- Sistemas distribuídos
- Interface entre usuários e sistemas computacionais
- Engenharia de software
- Banco de dados

Bibliografia

- [1] 3GPP. *GPRS adds packet-switched functionality to GSM networks*. Fev. de 2001. ENDEREÇO: <http://www.3gpp.org/technologies/keywords-acronyms/102-gprs-edge> (acesso em 12/09/2013).
- [2] Paulo Albuquerque e Cáudia Santos. *GPS para iniciantes*. Apostila do Mini Curso de GPS - XI Simpósio Brasileiro de Sensoriamento. Abr. de 2003.
- [3] Arduino. *Arduino Uno Specifications*. Set. de 2013. ENDEREÇO: <http://arduino.cc/en/Main/ArduinoBoardUno> (acesso em 12/09/2013).
- [4] G. Booch, J. Rumbaugh e I. Jacobson. *Unified Modeling Language User Guide*. 2nd Edition. Addison-Wesley Object Technology Series, 2005.
- [5] David Booth et al. *Web Services Architecture*. Fev. de 2004. ENDEREÇO: <http://www.w3.org/TR/ws-arch/> (acesso em 20/09/2013).
- [6] DENATRAN. *Estatísticas de roubos e furtos em 2011*. Maio de 2011. ENDEREÇO: <http://www.denatran.gov.br/publicacoes/publicacao.asp> (acesso em 12/02/2013).
- [7] Huascar Espinoza et al. “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>.
- [8] Erich Gamma. *Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos*. Bookman, 2000.
- [9] Dayani Adionel Guimarães. “Introdução as Comunicações Móveis”. Em: *Revista Telecomunicações - INATEL* 1 (1998).
- [10] Alfonso Diaz-Granados Marquez. “Sistema Antifurto de Veículos Automotivos”. Graduação. Curitiba: Núcleo de Ciências Exatas e de Tecnologia, Centro Universitário Positivo, 2006, p. 34.
- [11] Grant Martin et al. “Embedded UML: a merger of real-time UML and co-design”. Em: *Proceedings of the ninth international symposium on Hardware/software code-design*. 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>.
- [12] Grant Martin. “UML for Embedded Systems Specification and Design: Motivation and Overview”. Em: *Proceedings of the conference on Design, automation and test in Europe*. DATE '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 773–. ISBN: 0-7695-1471-5. ENDEREÇO: <http://dl.acm.org/citation.cfm?id=882452.874354>.

- [13] 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.
- [14] Peter Marwedel. *Embedded System Design: Embedded Systems Foundations of Cyber-Physical*. 2nd Edition. Springer, 2001.
- [15] Antônio Mendes. *Arquitetura de Software: Desenvolvimento orientado para arquitetura*. Editora Campus, 2002.
- [16] 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.
- [17] Trygve Reenskaug. *MVC XEROX PARC 1978-79*. Fev. de 1978. ENDEREÇO: <http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html> (acesso em 25/09/2013).
- [18] Sparkfun GPS Shield. *Retail Kit*. Abr. de 2013. ENDEREÇO: <https://www.sparkfun.com/products/10709> (acesso em 10/04/2013).
- [19] Wayne Wolf. *Computer as Components: Principles of Embedded Computing System Design*. 2nd Edition. Morgan Kaufmann Publishers, 2001.

Apêndice A

Anexos

A.1 Descrição dos casos de uso

Tabela A.1: Caso de Uso Manter Usuário

| | |
|---------------------|---|
| Nome do caso de uso | Manter Usuário |
| Sumário | Caso que descreve as etapas para manutenção de um usuário no sistema através da figura do Administrador, envolvendo as operação CRUD e associações com outros elementos do sistema. |
| Autor primário | Administrador |
| Atores secundários | Usuário comum |
| Precondições | Usuário com CPF válido. |
| Fluxo Principal | <p>1. Administrador deve se autenticar no sistema web.</p> <p>2. Acessar página de gerenciamento de Usuários.</p> <p>3. Escolher opção de cadastrar novo usuário.</p> <p>4. Preencher os dados com informações válidas.</p> <p>5. Confirmar ação e finalizar processo.</p> |
| Fluxo Alternativo | <p>(1) Credenciais inválidas</p> <p>a. O sistema retorna uma mensagem de erro informando o ocorrido e impossibilitando acesso sem autenticação.</p> <p>(3) Buscar Usuário</p> <p>a. Selecionar os filtros de busca e os valores para cada um deste.</p> <p>b. O usuário ou lista de usuários compatíveis com os filtros são retornados.</p> <p>(3) Apagar Usuário</p> <p>a. Selecionar usuários que se deseja remover através de um componente de seleção múltipla.</p> <p>b. O sistema retorna o nome dos usuários a serem apagados, solicitando confirmação.</p> <p>c. O administrador confirma a ação.</p> <p>(3) Atualizar dados do Usuário</p> <p>a. Selecionar usuário a sofrer o processo.</p> <p>b. Página de cadastro de usuário é preenchida com os dados atualmente persistidos.</p> <p>c. Preencher os dados com informações válidas.</p> <p>d. Confirmar ação e finalizar processo.</p> <p>(5) Falha na validação dos dados</p> <p>a. O sistema retornará mensagem informando quais campos possuem valor inválido.</p> |
| Pós-condições | Usuário estará cadastrado no sistema, podendo assim acessá-lo e executar as ações descritas em seus privilégios. |

| | |
|---------------------------|--|
| Requisitos não funcionais | Design minimalista e funcional da página de inserção de usuário, validação de valores dos campos, double-check para remoção de usuários. |
| Autor | Leandro Bentes |
| Data | 17/05/2013 |

Tabela A.2: Caso de Uso Manter ModVeicular

| | |
|---------------------|--|
| Nome do caso de uso | Manter ModVeicular |
| Sumário | Caso que descreve os processos de manutenção de um módulo veicular no sistema. O módulo veicular é um componente acoplado ao veículo que permite a este ser rastreado e bloqueado via web. |
| Ator primário | Administrador |
| Atores secundários | |
| Precondições | Existir módulo físico com a numeração serial única e válida. |
| Fluxo Principal | <p>1. Administrador deve se autenticar no sistema web.</p> <p>2. Acessar página de gerenciamento de Módulos Veiculares.</p> <p>3. Escolher opção de cadastrar novo módulo.</p> <p>4. Preencher os dados com informações válidas.</p> <p>5. Confirmar ação e finalizar processo.</p> |
| Fluxo Alternativo | <p>(1) Credenciais inválidas</p> <p>a. O sistema retorna uma mensagem de erro informando o ocorrido e impossibilitando acesso sem autenticação.</p> <p>(2) Buscar Módulo Veicular</p> <p>a. Selecionar os filtros e os valores para cada um deste.</p> <p>b. O módulo ou lista de módulos veiculares compatíveis com os filtros são retornados.</p> <p>(2) Atualizar dados do Módulo Veicular</p> <p>a. Buscar módulo veicular.</p> <p>b. Selecionar módulo a sofrer o processo, a página de cadastro deve surgir preenchida com os valores atuais de cadastro deste.</p> <p>c. Preencher os dados com informações válidas.</p> <p>d. Confirmar ação e finalizar processo.</p> <p>(2) Remover módulo veicular</p> <p>a. Selecionar equipamentos que se deseja remover através de um componente de seleção múltipla.</p> <p>b. O sistema retorna o número serial dos equipamentos a serem apagados, solicitando confirmação.</p> <p>c. O administrador confirma a ação.</p> <p>d. O sistema remove o(s) equipamento(s) desejados.</p> <p>(5) Falha na validação dos dados</p> <p>a. O sistema retornará mensagem informando quais</p> |

| | |
|---------------------------|---|
| Requisitos não funcionais | Design minimalista e funcional da página de Inserção de Equipamento, validação de valores dos campos. |
| Autor | Leandro Bentes |
| Data | 14/04/2013 |

Tabela A.3: Caso de Uso Associar Modulo-Usuário

| Nome do caso de uso | Associar Modulo-Usuário |
|---------------------|--|
| Sumário | Descrição da associação entre o equipamento e seu dono (um usuário cadastrado), possibilitando operações-chave como rastreamento e bloqueio. |
| Ator primário | Administrador |
| Atores secundários | Usuário comum |
| Precondições | Usuário e equipamentos envolvidos previamente cadastrados. |
| Fluxo Principal | <ol style="list-style-type: none"> 1. Administrador deve se autenticar no sistema web. 2. Acessar página de Associação de Equipamentos, a lista de equipamentos livres é exibida. 3. O administrador seleciona um equipamento, uma tela de pesquisa de usuários é mostrada. 4. O usuário para associação é mostrado. 5. O administrador deve confirmar a associação que é efetivada no sistema. |
| Fluxo Alternativo | <p>(1) Credenciais inválidas</p> <ol style="list-style-type: none"> a. O sistema retorna uma mensagem de erro informando o ocorrido e impossibilitando acesso sem autenticação. <p>(3) Caso executado a partir do cadastro de Usuário.</p> <ol style="list-style-type: none"> a. A tela de pesquisa de usuários não é mostrada, uma vez que a associação será feita com o usuário que está sendo cadastrado. |

| | |
|---------------------------|--|
| Pós-condições | O usuário que se associou ao equipamento pode rastreá-lo, e bloquear veículo no qual ele está instalado. Equipamento não poderá ser associado a outro usuário sem que se remova a atual associação. |
| Requisitos não funcionais | Não definidos ainda |
| Autor | Leandro Bentes |
| Data | 10/04/2013 |

Tabela A.4: Caso de Uso Atualizar Próprio Cadastro

| | |
|---------------------|---|
| Nome do caso de uso | Atualizar Próprio Cadastro |
| Sumário | Caso que descreve a atualização dos próprios dados, feita por um usuário acessando o sistema. |
| Ator primário | Usuário comum, Administrador |
| Atores secundários | |
| Precondições | |
| Fluxo Principal | <ol style="list-style-type: none"> 1. Usuário deve se autenticar no sistema web. 2. Acessar página Meus Dados. 3. O formulário é preenchido com os dados atualmente persistidos. 4. O usuário altera os campos desejados e aciona opção salvar. 5. O sistema valida os valores e os dados são persistidos. |

| | |
|---------------------------|--|
| Fluxo Alternativo | <p>(1) Credenciais inválidas</p> <p>a. O sistema retorna uma mensagem de erro informando o ocorrido e impossibilitando acesso sem autenticação.</p> <p>(5) Falha na validação dos dados</p> <p>a. O sistema retornará mensagem informando quais campos possuem valor inválido, para que o usuário possa corrigir facilmente em caso de engano.</p> |
| Pós-condições | Modificação nos dados iniciais do Usuário. |
| Requisitos não funcionais | Design minimalista e funcional da página de Inserção de Usuário, validação de valores dos campos. |
| Autor | Leandro Bentes |
| Data | 14/04/2013 |

Tabela A.5: Caso de Uso Manter Veículos

| | |
|---------------------|---|
| Nome do caso de uso | Manter Veículos |
| Sumário | Caso que descreve as etapas para manutenção de veículos do usuário no sistema, envolvendo as operações CRUD. |
| Ator primário | Usuário comum |
| Atores secundários | |
| Precondições | Veículo com placa e código de chassi válido. 1. Usuário deve se autenticar no sistema web. 2. Acessar página de gerenciamento de Veículos. 3. Escolher opção de cadastrar novo veículo. 4. Preencher os dados com informações válidas. 5. Confirmar ação e finalizar processo. |
| Fluxo Principal | |

| | |
|-------------------|---|
| | <p>(1) Credenciais inválidas</p> <p>a. O sistema retorna uma mensagem de erro informando o ocorrido e impossibilitando acesso sem autenticação.</p> <p>(3) Buscar Veículo</p> <p>a. Selecionar os filtros de busca e os valores para cada um deste.</p> <p>b. O veículo ou lista de veículos compatíveis com os filtros são retornados.</p> <p>(3) Apagar Veículo</p> <p>a. Selecionar os veículos que se deseja remover através de um componente de seleção múltipla.</p> <p>b. O sistema retorna marca, modelo e placa dos veículos a serem apagados, solicitando confirmação.</p> <p>c. O usuário confirma a ação.</p> <p>(3) Atualizar dados do Veículo</p> <p>a. Após uma busca, selecionar o veículo a sofrer o processo.</p> <p>b. Página de cadastro de veículo é preenchida com os dados atualmente persistidos.</p> <p>c. Preencher os dados com informações válidas.</p> <p>d. Confirmar ação e finalizar processo.</p> <p>(5) Falha na validação dos dados</p> <p>a. O sistema retornará mensagem informando quais campos possuem valor inválido, para que o usuário possa corrigir facilmente em caso de engano.</p> <p>b. Caso o veículo a ser cadastrado já exista no banco de dados, o sistema informará esta situação com mensagem apropriada.</p> |
| Fluxo Alternativo | |

| | |
|---------------------------|---|
| Pós-condições | Veículo estará cadastrado no sistema, podendo ser associado a algum equipamento para quer o rastreio e/ou bloqueio sejam permitidos. Em caso de remoção, o módulo associado ao veículo será marcado como livre, somente podendo sofrer bloqueio e rastreamento caso seja associado a um novo. |
| Requisitos não funcionais | Design minimalista e funcional da página de inserção de veículo, validação de valores dos campos, double-check para remoção de veículo. |
| Autor | Leandro Bentes |
| Data | 19/05/2013 |

Tabela A.6: Caso de Uso Listar Módulos

| | |
|---------------------|---|
| Nome do caso de uso | Listar Modulos |
| Sumário | Caso que descreve os processos de manutenção dos módulos veiculares pertencentes ao usuário. |
| Ator primário | Usuário |
| Atores secundários | |
| Precondições | Usuário cadastrado e equipamentos associados a ele. |
| Fluxo Principal | <ol style="list-style-type: none"> 1. Usuário deve se autenticar no sistema web. 2. Acessar página Meus Módulos ou nome correspondente. 3. Todos os módulos pertencentes ao usuário são retornados, podendo associá-lo ou desassociá-lo de um veículo. 4. Usuário filtra o resultado para encontrar o(s) módulo(s) desejado(s). |

| | |
|-------------------|--|
| Fluxo Alternativo | <p>(1) Credenciais inválidas</p> <ul style="list-style-type: none">a. O sistema retorna uma mensagem de erro informando o ocorrido e impossibilitando acesso sem autenticação. <p>(3) Detalhar módulo</p> <ul style="list-style-type: none">a. Usuário filtra o resultado para encontrar o(s) módulo(s) desejado(s).b. Selecionar opção Detalhar Módulo. <p>(3) Usuário sem módulo pertencente a ele.</p> <ul style="list-style-type: none">a. O sistema retornará mensagem informando que o usuário não possui módulos cadastrados. <p>(4) Filtro sem correspondência</p> <ul style="list-style-type: none">a. Caso o usuário aplique um filtro no resultado que não encontre qualquer módulo correspondente, o sistema informará que não existem resultados para o filtro aplicado. |
|-------------------|--|

| | |
|---------------------------|--|
| Pós-condições | Possibilidade de associar módulo a um veículo livre. |
| Requisitos não funcionais | |
| Autor | Leandro Bentes |
| Data | 15/04/2013 |

A.2 Códigos, esquemas e diagramas

Um dos principais algoritmos gerados neste projeto foi o firmware do módulo veicular embarcado. Utilizando uma combinação de bibliotecas de interface com os shields e funções implementadas de acordo com os diagramas de atividades das figuras 3.17 e 3.18 foi gerada a rotina embarcada.

Os códigos A.2.1, A.2.2, A.2.3 e A.2.4 apresentam o firmware em linguagem C/C++ gerado neste projeto.

```

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 = "lm25ttd.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 numErrors=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(cellsetUpPDPContext(&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 }
```

Código A.2.1: Código do Módulo Embarcado Parte 1

```

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     return false;
17 }
18
19
20
21 void rebootGSMPprocedure()
22 {
23     digitalWrite(REBOOT_PIN, LOW);
24 }
25
26 void signalizeError()
27 {
28     numOfErrors++;
29     if(numOfErrors>MAX_NUM_ERRORS)
30     {
31         rebootGSMPprocedure();
32     }
33 }
34
35 boolean sendMessageToServer(String *request, byte *connectionId)
36 {
37     if(cell.checkSocketStatusTCP())
38     {
39         numOfErrors=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\nHost: ";
8      request += HOST;
9      request += "\nConnection: keep-alive";
10     request += "\nUser-Agent: ";
11     request += (USER_AGENT+"\n");
12     request += "Accept: application/json\n";
13     request += "Content-Length: ";
14     request += parameters.length();
15     request += "\n";
16     request += "Content-Type: application/json\n\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 A.2.4: *Código do Módulo Embarcado Parte 4*

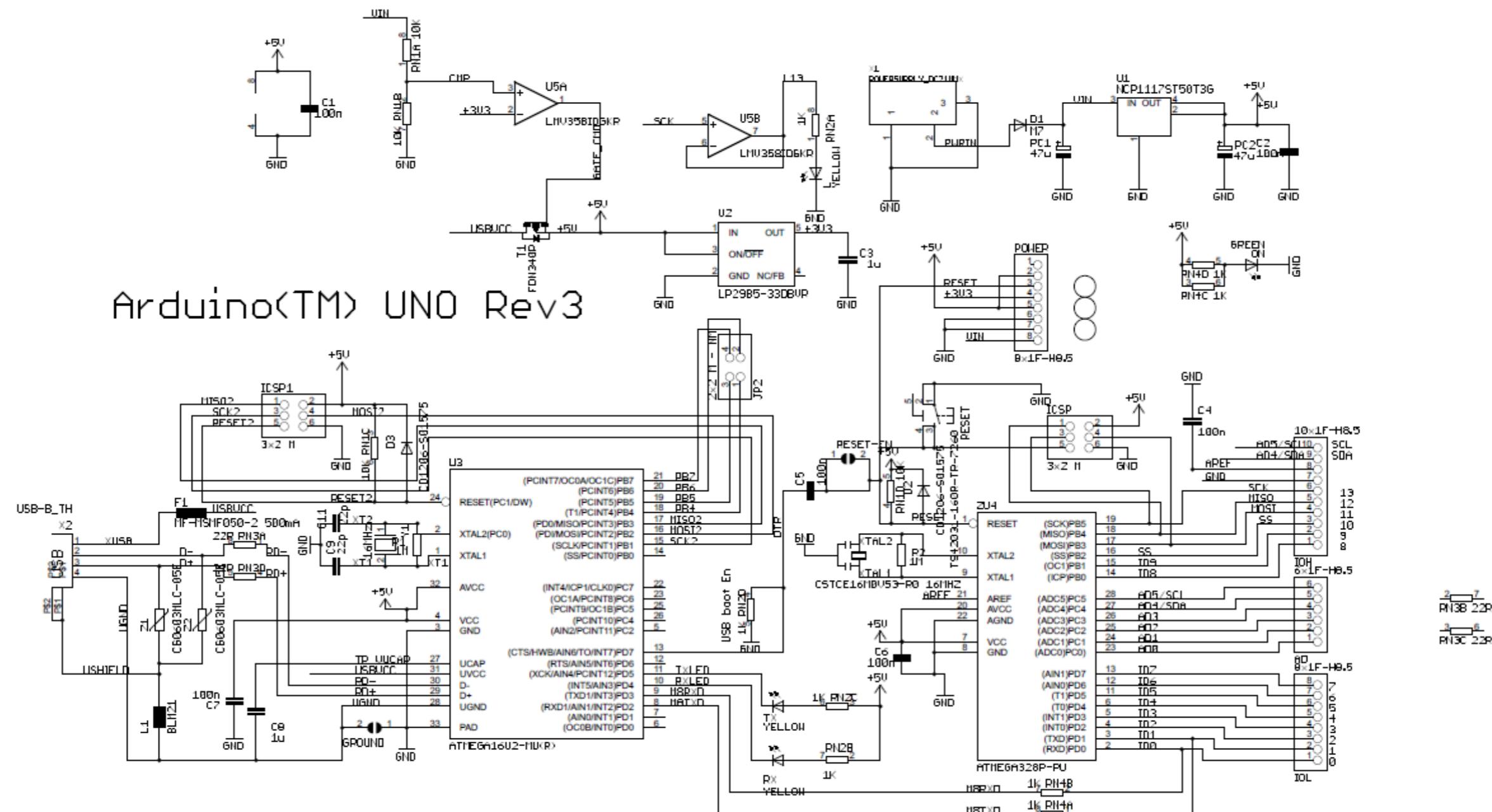


Figura A.1: Esquema elétrico da placa Arduino Uno

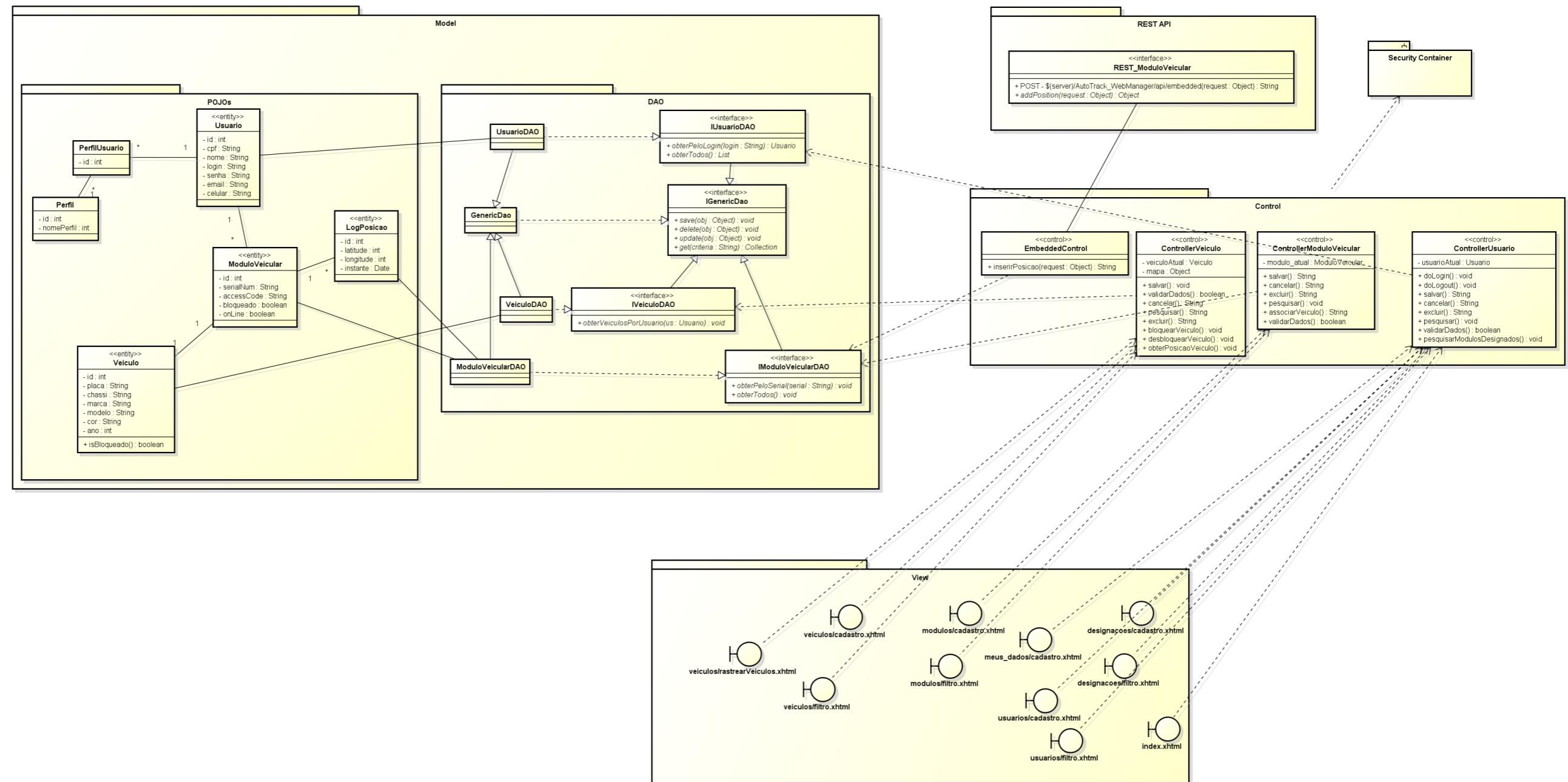


Figura A.2: Modelo de classes do WebManager

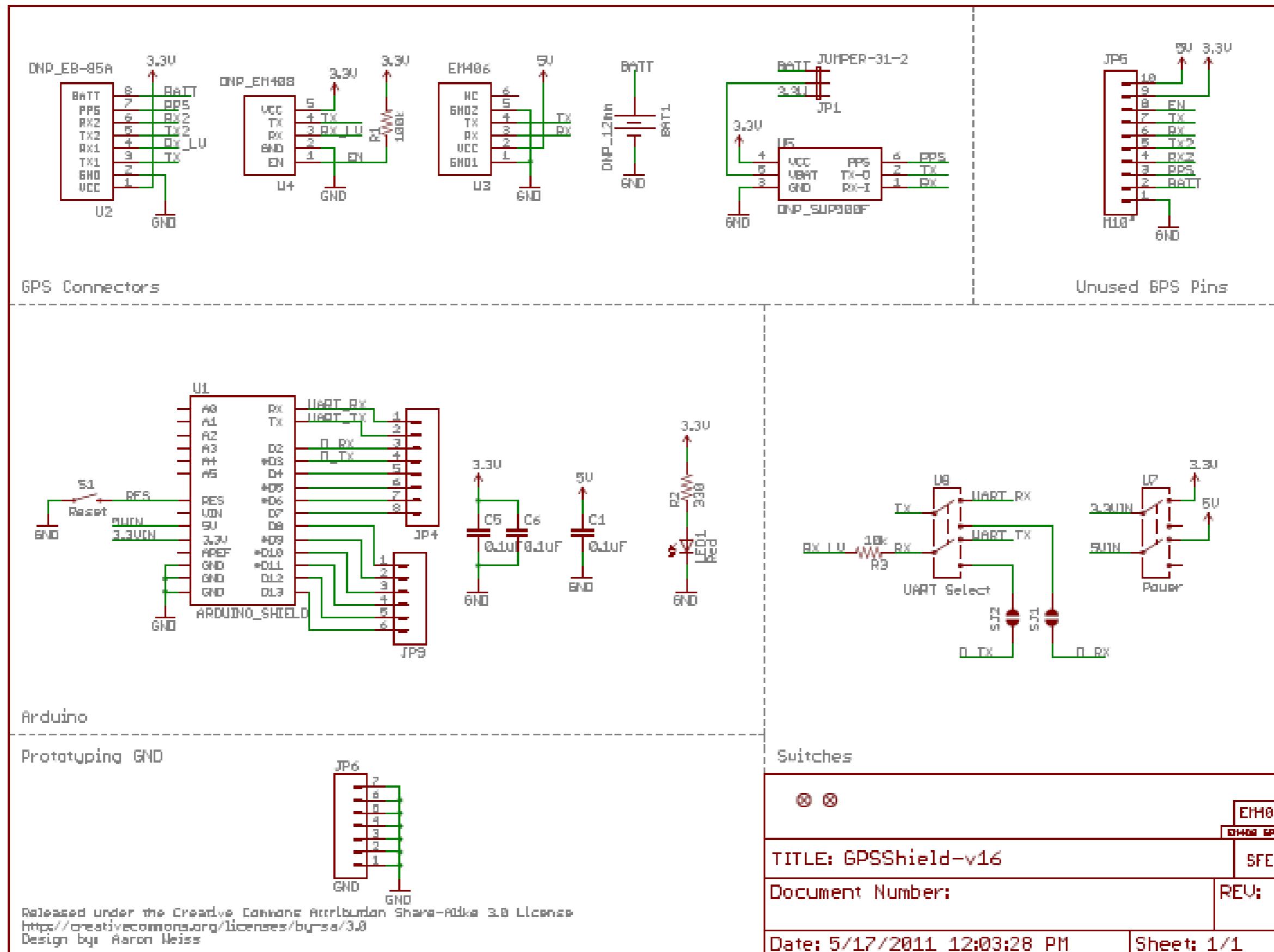


Figura A.3: Esquema elétrico do GPS Shield