Sistema de Gerência Remota de Alarmes

Luiz Fernando da Silva[1], F.C. Jabour[1]

[1]Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais - Campus Juiz de Fora

lfernando-silva@hotmail.com,filippe.jabour@ifsudestemg.edu.br

Abstract. *This paper describes and specifies the construction of a remote management system of automotive alarms based on Internet of Things (IoT) concept and asynchronous and non-blocking Web technologies. Furthermore, it describes a reusable protocol in other telemetry applications and remote interventions. The built system was efficient in relation to the work domain, through quick, reliable and reduced data traffic triggers, the latter also is due to appropriate choice of serial written procedure on the associated embedded system. Thus, it constitutes an alternative to the traditional model of alarms based on the use of radiofrequency waves (UHF).*

Resumo. *Este trabalho descreve e especifica a construção de um sistema de gerenciamento remoto de alarmes automotivos baseado no conceito de Internet das Coisas (Internet Of Things – IoT)* e de *tecnologias Web assíncronas e não bloqueantes. Além disso, descreve um protocolo reutilizável em outras aplicações de telemetria e intervenção remota. O sistema construído mostrou-se eficiente em relação ao domínio do trabalho, através de acionamentos rápidos, confiáveis e com tráfego reduzido de dados, este último também graças a escolha adequada do procedimento de escrita serial do sistema embarcado associado. Com isso, consiste uma alternativa ao modelo tradicional de alarmes, baseados no uso de ondas de radiofrequência (UHF).*

1. Introdução

1.1. Contextualização

Nos últimos quinze anos, o Brasil passou por diversas mudanças, dentre elas a de perfil socioeconômico e de consumo. Um dos impactos é observado no aumento de mais de 100% no número de veículos desde 2003 [DENATRAN 2014]. Sendo assim, a segurança veicular se torna cada vez mais importante, já que quase 230 mil veículos foram roubados no país em 2013 por exemplo, sendo um dos tipos de roubo mais frequentes no país [Gomes 2015].

A proteção aos próprios veículos significa um custo de manutenção, através de seguros e sistemas de alarme sonoros. Os seguros, no Brasil, estão entre os mais caros do mundo, preço que está relacionado a um índice conhecido como "índice de furto", que trata da probabilidade de um carro ser roubado/furtado [Almeida 2015]. Já os alarmes automotivos, normalmente mais baratos e de fácil instalação, são geralmente compostos por um controle que emite um sinal UHF em 433MHz, conforme resolução da ANATEL, para um dispositivo dentro do veículo, podendo assim travar ou destravar as portas, sensores, etc. Se algum compartimento do carro, ligado ao alarme, for

modificado sem que o alarme esteja desligado, ele dispara com sinais sonoros e luminosos para alertar o proprietário. No entanto, nem sempre ele está perto para ser notificado. A figura 1 mostra um esquema genérico de um alarme automotivo comum. Destaca-se que as flechas representam o fluxo de dados no sistema.

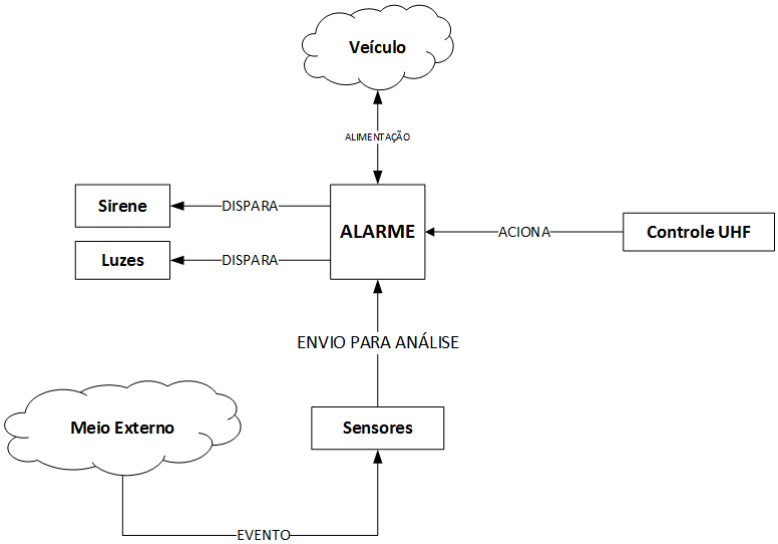

Figura 1. Esquema genérico de um alarme automotivo comum.

O uso da radiofrequência constitui uma grande fragilidade, pois é relativamente fácil provocar interferência no acionamento através de "bloqueadores de radiofrequência", conhecidos como "chapolins" ou "*jammers*". Estes aparelhos possuem várias aplicações, entre elas, podem embaralhar ou até copiar o sinal emitido, já que se conhece a frequência de operação.

1.2. Visão Geral deste Trabalho

A proposta deste trabalho é a criação do Sistema de Gerência Remota de Alarmes (SGRA), um sistema auxiliar que não se baseia na frequência UHF para ativação do alarme, mas na Internet e na rede de telefonia, já que operam em frequência superior a 433MHz e, em tese, não estariam sujeitas à mesma fragilidade. Além disso, aplica-se ao trabalho o conceito de "Internet das Coisas", com o objetivo de criar uma camada intermediária composta por um sistema web que fará a gerência dos acionamentos de maneira mais segura. Sendo assim, basta que o usuário acesse o sistema, realize a autenticação e acione o alarme através de sua página, dispensando a utilização do controle nativo.

A figura 2 atualiza o esquema da figura 1 com disposição do SGRA, em azul, em relação ao alarme automotivo. Deve-se observar que o controle UHF não faz parte da nova disposição do sistema, bem como que o sistema embarcado não interfere na política de alarme disparado, isto é, não importa quais sensores estão sendo monitorados, tampouco como o alarme entende que 'deve disparar'. Nestes casos, o sistema embarcado do SGRA apenas receberá a saída que também será enviada aos faróis e à sirene, por exemplo. De forma análoga, o sistema embarcado estará ligado diretamente ao módulo de acionamento do alarme, em que a entrada deste módulo é a saída do processamento resultante do acionamento na "interface do usuário".

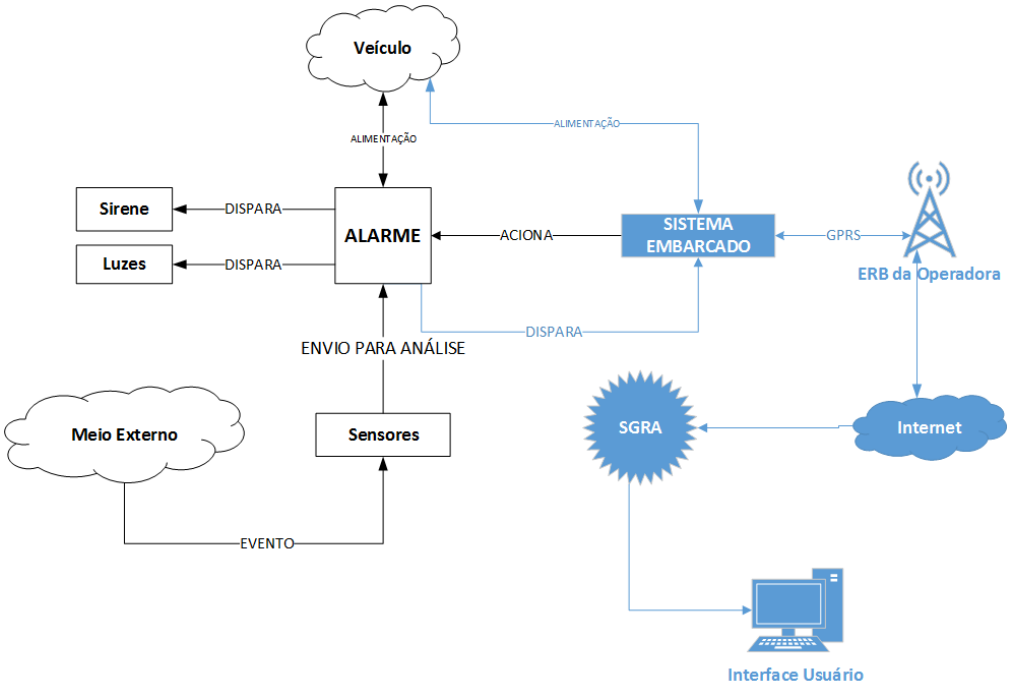

Figura 2. Esquema genérico de um alarme associado ao SGRA.

Entre os objetivos do SGRA, estão: gerência do acesso aos veículos, ou seja, o destravar o alarme mediante autenticação; acionamento do alarme via Internet, sem necessidade de estar próximo ao veículo; recebimento de notificações quando o alarme disparar, e; visualização de estatísticas que forem pertinentes, como data e hora dos acionamentos. A gerência do acionamento pela aplicação web visa dificultar que se ative o alarme sem autorização, já que não poderia ocorrer simplesmente copiando a chave, e mesmo esse procedimento de cópia fica amplamente comprometido pelas políticas de segurança da rede de telefonia. O acionamento via Internet é interessante particularmente quando existe dúvida se o alarme foi ativado, podendo conferir e acioná-lo a distância. O recebimento de notificações é importante aliado do alarme sonoro, em que ao disparar, o dispositivo embarcado também envia a notificação para o proprietário, mesmo estando distante do veículo, possibilitando e melhorando medidas de proteção ao veículo. Por fim, estatísticas podem ter diversas utilidades, que irão variar de acordo com o contexto. O importante neste caso é ter a ciência de que estes dados, trabalhados numa plataforma web, podem utilizar recursos que vão de processamentos matemáticos a utilização de recursos estatísticos, probabilísticos e outros que sejam amplamente usados em processos gerenciais. Tudo isso, usando apenas os dados e pouco dependentes da estrutura de hardware suas limitações.

O sistema proposto não substitui o sistema de alarme em si, mas modifica sua metodologia original de funcionamento. Além disso, utiliza um tipo de protocolo que pode ser reutilizado e expandido para várias aplicações em automação e telemetria que necessitem de mobilidade e/ou intervenção remota.

2. Metodologia

O SGRA se encaixa no conceito de "Internet das Coisas", "*Internet Of Things*" ou simplesmente IoT, termos que se referem a sistemas de dispositivos, interconectados ou não, conectados a sistemas na web [Nascimento 2015]. Um sistema IoT é composto por

pelo menos quatro elementos [Luz e Humenhuk 2015]: a "coisa", o sistema embarcado encarregado de receber as informações do ambiente e enviá-las; a conectividade, o meio de tráfego dos dados; os dados coletados pelo sistema embarcado, e; a análise, que é a utilização desses dados por outros sistemas e ferramentas.

Para essa proposta de sistema, as especificações de software devem prover artefatos que permitam reproduzir, manter e ampliá-los, sendo base para projetos de software. Elas envolvem desde a concepção até as arquiteturas do sistema [Lima 2012, Sommerville 2003] até o domínio da aplicação [Silva e Videira 2001].

2.1.1. Sistema Embarcado

Um sistema embarcado [Moroz 2012] é um sistema computacional dedicado e de tamanho reduzido, sendo a interface entre o meio externo e a "coisa" da IoT. Neste trabalho, ele é composto por uma composição de circuitos feita com um Arduino Uno R3 e com uma *shield* Tinysine com o módulo SIM900 da SimComm.

O Arduino [McRoberts 2011] é um circuito integrado livre resultado de um projeto iniciado em 2005, conhecido por sua fácil programação, cujos códigos são conhecidos como *sketchs,* e seu amplo uso para fins educacionais e projetos de automação e robótica independentes [Margolis 2011, McRoberts 2011]. Ele é capaz de interagir com outros dispositivos, como sensores e atuadores [Margolis 2011], que modificam o formato da energia [Medeiros 2007] e possibilitam o manuseio dessas informações ao transformá-las em dados [Margolis 2011].

Contextualmente, não há sentido em um sistema embarcado que não se comunique com outros dispositivos, componentes ou, no caso de IoT, com uma API (*Application Programming Interface),* mecanismo que fará o devido tratamento dos dados para serem utilizados pelo sistema. Essa comunicação deve ser feita através de um módulo de conectividade [Luz e Humenhuk 2015, Moroz 2012]. Neste caso, a utilização de outro circuito com fácil integração é interessante, já que a alternativa seria um projeto de circuito integrado que conectasse todos os componentes demandados, cuja conveniência deve ser avaliada. Assim, um circuito do tipo *shield* é capaz de simplificar essa interação, já que é construído especialmente para ser acoplado a um circuito, que neste caso é o Arduino Uno R3, estendendo as funcionalidades deste de forma simplificada.

Assim, faz-se presente a *shield* Tinysine GSM/2G, que realizará a comunicação do Arduino Uno R3 com a rede de telefonia, através do módulo de processamento Sim900. O Sim900 é um núcleo de processamento GSM/GPRS *quad-band* (850MHz, 900MHz, 1800MHz e 1900MHz) que provê mecanismos para transmissão de dados via GPRS (*General Packet Radio Service*), que são os comandos AT [SimComm 2009]. Esses comandos são padronizados e definidos na documentação oficial do Sim900, bem como as respostas esperadas, sendo enviados pela interface serial ou UART (*Universal Asynchronous Receiver/Transmitter*) através dos pinos 0 e 1 do Arduino, que representam leitura e escrita, respectivamente [Margolis 2011].

2.1.2. Rede de Telefonia Móvel (GSM)

Por rede de telefonia móvel, entende-se o sistema interligado de comunicações sem fio numa determinada área de cobertura, constituído de rádios móveis, telefones celulares e antenas conhecidas como estações rádio base, ou ERB's [Medeiros 2007].

Normalmente, a rede de telefonia, especialmente no Brasil, é observada com certa desconfiança quanto à eficiência, uma vez que são inúmeras as variações de cobertura, qualidade de sinal e largura de banda por conta da localização dessas ERB's. Comumente, os usuários associam a qualidade deste serviço ao desempenho que possuem em seus *smartphones*. No entanto, o advento desses *smartphones* forçou uma melhoria no serviço e a possibilidade de uma transmissão razoável de dados, não só para operações telefônicas, mas para outros serviços como *e-mails,* mensagens instantâneas, reprodução de áudio/vídeo, dentre outras [Molisch 2011].

O contexto atual do país é de uma cobertura de celular relativamente alta nas principais cidades, como pode ser visto no serviço OpenSignal, em www.opensignal.com, que fornece dados baseados na quantidade de usuários conectados às ERB's. A figura 3 representa a medição feita para a cidade de Juiz de Fora - MG, mostrando índices relativamente positivos para a utilização de 2G e 3G.

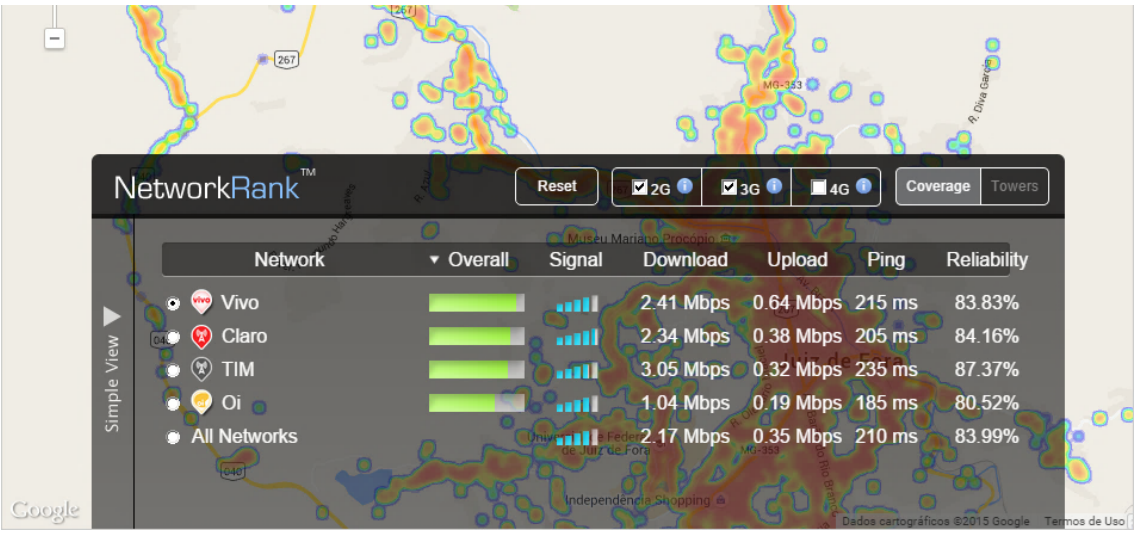

Figura 3. Índices de 2G/3G das principais operadoras de telefonia em Juiz de Fora - MG.

O uso da rede de telefonia é particularmente conveniente no contexto IoT por uma série de razões como a mobilidade, a cobertura e a possibilidade de atender mais clientes do que outras modalidades de conectividade sem fio comuns, como WLAN [Molisch 2011]. Além disso, não exige a implantação de uma infraestrutura de rede, bastando a aquisição de um plano de telefonia adequado ao universo da aplicação e um sistema embarcado compatível. O sistema IoT deve prezar sempre por um tráfego otimizado de dados nessa rede, implicando numa redução de custo ao adotar-se uma infraestrutura mais específica.

No quesito de segurança, a rede de telefonia é relativamente segura, já que utiliza criptografias e processos de autenticação. No entanto, é possível interferir em seu funcionamento utilizando ataques do tipo *man-in-the-middle* [Molisch 2011], ataques que basicamente conseguem interceptar dados de uma transmissão por GPRS, problemas que podem ser atenuados com a adoção de políticas de segurança bem definidas no escopo da aplicação web. No entanto, esses ataques devem ocorrer em condições muito específicas, o que não inviabiliza o uso da rede 2G. Já as redes 3G e 4G possuem mecanismos melhores de impedir esse tipo de ataque.

2.1.3. Dados e Análise

Outra parte importante da IoT é a geração e análise de dados, resultantes de coletas do sistema embarcado e da plataforma Web, que constituem informação implícita [Elmasri e Navathe 2005] e também dados de entrada em um sistema IoT. Neste caso, é necessário um sistema que processe e persista dados com alta performance.

O NodeJS é uma tecnologia criada em 2009 por uma equipe liderada por Ryan Dahl, baseada no *Javascript V8 Engine* do Google, usada no Google Chrome. Logo, a codificação para NodeJS é em linguagem de programação Javascript, que segue a referência da linguagem ECMAScript tal como outras linguagens de script, e possui características como o trabalho em contextos assíncronos e tipagem fraca, por exemplo [Dayley 2014].

Entender o potencial do NodeJS aplicado a IoT requer o entendimento dos contextos síncronos e assíncronos, bloqueantes e não bloqueantes de processamento. As operações bloqueantes, como consultas ao banco de dados, envio de e-mails, dentre outras, são consideradas críticas, já que impedem que outras tarefas sejam executadas durante seu processamento, isto é, bloqueiam a *thread*, em que estão sendo executadas e deixam as demais operações em espera. Entende-se por *thread* [Tanembaum 2009, Dayley 2014] o mecanismo com o qual o Sistema Operacional divide um processo em vários subprocessos "quase-paralelos", para que um destes que seja bloqueante não impeça a execução de outro.

Se uma operação bloqueante nunca acabar, como em casos de ciclos de repetição infinitos, ou ainda em operações das quais necessariamente se espera uma resposta para executar o próximo processo, o sistema pode entrar em um estado conhecido por "*deadlock*" [Pereira 2013], em que as demais operações nunca serão executadas naquela *thread*. Neste caso, uma solução é a construção de sistemas *multithreading*, que são capazes de executar várias operações simultaneamente [Tanembaum 2009], mas que são pouco escaláveis em trocas de contexto e sincronização, além de serem mais complexos de se gerenciar e consumirem mais recursos computacionais [Schmidt 2000], o que pode representar um aumento no custo de hospedagens e servidores.

De qualquer forma, os problemas começam a surgir com o aumento da quantidade de requisições, simultâneas ou não, que devem ser tratadas eficientemente, especialmente no caso de IoT. Para essas situações, o contexto assíncrono/não bloqueante é uma alternativa mais eficiente. O NodeJS possui assincronismo nativo, graças à implementação de um padrão voltado para arquitetura de sistemas conhecido como *Reactor* [Casciaro 2014, Schmidt 200] em seu núcleo, que tornam a aplicação orientada a eventos [Schmidt 2000]. Esses eventos são associados a estruturas que manipulam eventos, os *event handlers*, que equivalem aos *callbacks* e *promises* do Javascript e são colocadas numa grande fila circular de eventos, o *event loop* [Pereira 2013].

O NodeJS permitiu, na prática, executar Javascript do lado do servidor e a criação de aplicações REST (*Representational State Transfer*), utilizando os métodos HTTP e mecanismos de rotas [Mardan 2014, Saudate 2013], com acesso a bancos de dados e a outras API's. No entanto, é importante ter uma base de dados que acompanhe esse desempenho, caso do MongoDB, uma alternativa interessante para o modelo IoT.

Ele é um banco de dados não-relacional, orientado a documentos na notação JSON [MongoDB 2015] que permite trabalhar com objetos complexos e estruturas de dados como vetores (*arrays*), o que é mais próximo da aplicação. Neste caso, os *documents* (documentos) são análogos aos registros, e as *collections* (coleções), às tabelas. A figura 4 mostra um *document* armazenado no MongoDB, na estrutura JSON, que representa os dados relativos a um usuário do SGRA, seus respectivos veículos e acionamentos de alarme.

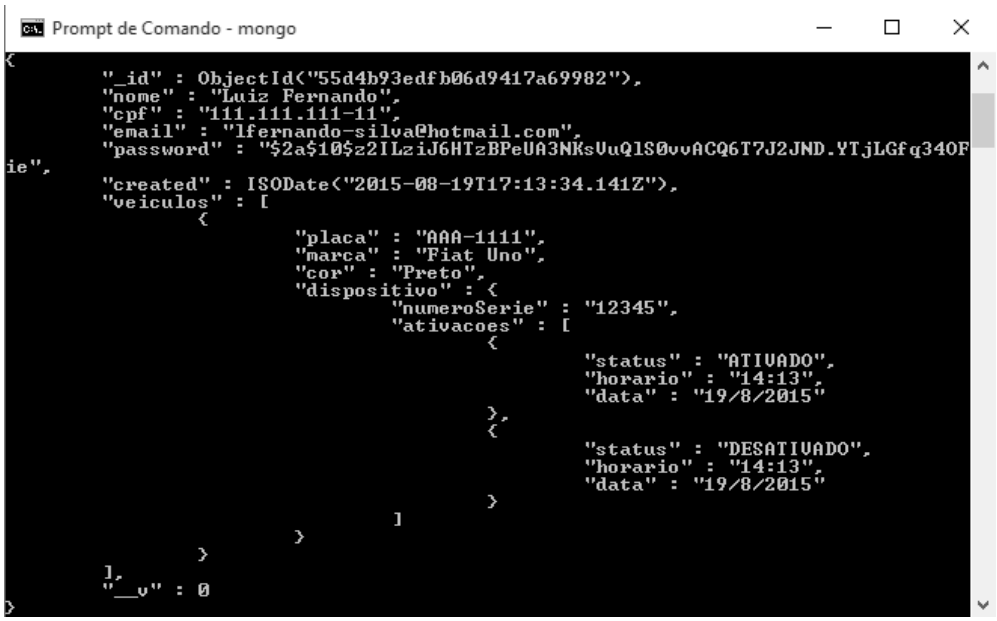

Figura 4. Documento JSON armazenando dados do usuário, do veículo e dois estados do alarme do veículo no protótipo em uso.

A grande flexibilidade no armazenamento de dados é bastante útil com NodeJS, já que o Javascript é de tipagem fraca, ou seja, não é necessário determinar de antemão qual é o tipo das variáveis (Inteiro, Número Real, String, Booleano, entre outros) a serem utilizadas, como em outras linguagens como Java e C++. Ao contrário dos bancos relacionais, o MongoDB é *schemaless*, isto é, possui estruturas flexíveis de persistência. Inclusive, permite o armazenamento de funções em Javascript, ideia análoga às *triggers* do mundo relacional. No entanto, não suporta *transactions,* que são execuções conjuntas de comandos [Beighley 2007], nem *rollbacks*, que é o cancelamento dessas transações [Beighley 2007], e embora o recente suporte à *joins,* o mais comum é trabalhar com esse banco na forma *denormalizada,* isto é, com acesso atômico aos campos através da notação JSON e "*pre-joins"* [MongoDB 2015].

Sua alta performance e alta escalabilidade são graças ao paradigma NoSQL, que tem sido muito utilizado em *big datas,* grandes bases de dados com alto tráfego [Akerkar 2014], em que não só é necessário um grande espaço de armazenamento, mas também flexibilidade e performance.

2.1.4. Javascript e Orientação a Objetos

Parte essencial do desenvolvimento de um sistema está na adoção das melhores práticas de desenvolvimento para facilitar a manutenibilidade, a legibilidade, a reutilização de código e otimização de seu funcionamento, além da redução da possibilidade de falhas, atingindo uma boa relação custo-benefício [Sommerville 2003]. Essas melhores práticas

normalmente estão associadas a princípios de desenvolvimento [Casciaro 2014] como códigos modularizados, com poucas responsabilidades, pouco interdependentes, entre outros, representados por siglas como KISS, SOLID e DRY, além de padrões de projeto. Assim, uma forma de consolidar esses princípios é através da orientação a objetos (OO), associada a políticas de documentação e modelagem. No entanto, a *filosofia* do NodeJS passa pelas particularidades do Javascript ES5 (versão adotada neste trabalho por ser ainda amplamente adotada na bibliografia e em milhares de projetos Javascript).

No paradigma do ES5, é possível se manipular objetos Javascript sem restrições de estrutura [Zakas 2014], assim como vários recursos da OO de forma diferente, como herança, polimorfismo e classes, que são utilizados na forma de padrões de projeto (especialmente os de criação de objetos) [Casciaro 2014, Zakas 2014, Stefanov 2010].

As particularidades da linguagem trazem padrões como *Module* e *Callback* [Casciaro 2014]. Os *modules* são scripts externos que podem incrementar a aplicação, em que correspondem às bibliotecas, frameworks e módulos propriamente ditos, sendo usados de maneira uniforme pelo NodeJS [Pereira 2013]. O *module* inclui um arquivo chamado *package.json*, um descritor que registra os metadados do módulo, suas dependências [Dayley 2014] e suas respectivas versões. Esses módulos são gerenciáveis pelo gerenciador de dependências do NodeJS, o *Node Package Manager* (NPM), que lê esses arquivos package.json e importa todos as dependências automaticamente. Alguns módulos fazem parte do núcleo do NodeJS (como *Path* e *Net*).

Já os *callbacks* são a materialização dos *event handlers* do padrão arquitetural *Reactor*, que são funções que propagam o resultado de suas operações através de eventos [Pereira 2013]. Com isso, as funções não *retornam* o resultado de suas operações, mas *propagam* para a próxima função. Trabalhando cooperativamente com o padrão *Reactor*, o padrão *callback* realiza um encadeamento dinâmico dos eventos, o que acaba por formar um *event loop* assíncrono.

3. Estruturação e Prototipação do SGRA

O sistema protótipo é uma aplicação de duas camadas: uma REST e outra TCP/IP. A primeira corresponde a interface com o usuário, enquanto a segunda, com o sistema embarcado conectado ao alarme, este que terá sua ideia abstraída por simples acendimentos de *led* e pressionamento de botão, conforme será explicado.

3.1. Especificação do Sistema

As características atuais da linguagem Javascript se alinham a vários recursos das especificações técnica e funcional, bem como da OO. Porém, a abordagem caracterizada pelo uso do Javascript ES5 associado a um modelo de banco de dados não-relacional (MongoDB) podem inviabilizar parte dessa especificação. Isto porque boa parte do NodeJS está voltado ao *comportamento* e ao *desempenho* da aplicação [Pereira 2013], considerando que a codificação na forma do ES5, isto é, o uso de *callbacks*, não só modifica a forma com que se trabalha com objetos [Stefanov 2010, Zakas 2014], mas com que se trabalha com o domínio da aplicação [Silva e Videira 2011]. Esta última é orientada a eventos e não baseada em classes, mas em módulos [Casciaro 2014, Pereira 2013]. Tudo isso se agrava ao fato de utilizar-se uma base de dados não relacional que preza por modelos de dados *denormalizados* e pré-associados [MongoDB 2015].

Para se entender melhor a dimensão do problema, por exemplo, pode-se observar que módulos criados *não* são classes: possuem vários dos comportamentos de classe, exceto o fato de que são quase sempre *singletons*, instâncias únicas de um objeto que representa uma entidade [Casciaro 2014], chamados com o método *require*, caracterizando o padrão *module* do NodeJS [Casciaro 2014]. Sendo assim, atributos e métodos não pertencem a uma classe, mas a um objeto, como quase tudo em Javascript [Zakas 2014]. Isso inviabiliza uma representação UML de classes de um software feito na forma do ES5, bem como inviabiliza algumas representações baseadas em *instâncias* de uma classe, já que os objetos Javascript não são instâncias, mas "cópias" de um objeto genérico ou literal, padrão Javascript conhecido como *prototypical inheritance*, bastante similar ao padrão GoF *Prototype* [Mardan 2014]. A figura 5 mostra um *module* que representa a entidade "usuário" que será utilizada no sistema. É possível perceber, por exemplo, a existência de um atributo "veiculos", que é um *array* de objetos, contrariando o princípio da normalização, que exigiria que existisse uma entidade do tipo veículo em outra classe ou módulo, e esta estaria ligada à primeira através de chave estrangeira. Na forma denormalizada, o sistema apenas manipula uma entidade, em que seus atributos contêm as outras entidades [MongoDB 2015]. Pode-se perceber, ainda, um *module* externo sendo chamado à estrutura, o módulo *mongoose,* através do método *require*.

```
user.js C:\Users\Luiz\Documents\Visual Studio 2013\Projects\Sistema Alarme\Sistema Alarme\models
1  var mongoose = require('mongoose');
2  var Schema = mongoose.Schema;
3
4
5  var userSchema = new Schema({
6      nome: { type: String, required: "Insira seu primeiro nome" },
7      cpf: {type:String, required: "Insira seu CPF" },
8      email: {type:String, required: "Insira seu email" },
9      password: { type: String, required: "Insira a senha" },
10     veiculos: {type: Array, default: []},
11     created: {type: Date, default: Date.now}
12 });
13
14 //validar email, chamada a função definida no user-service.js
15 userSchema.path('email').validate(function (value, next) {
16     var userService = require("../services/user-service");
17     userService.findUser(value, function (err, user) {
18         if (err) {
19             console.log(err);
20             return next(false);
21         }
22         next(!user);
23     });
24 }, "Email já existente!");
25
26 var User = mongoose.model("User", userSchema);
27
28 module.exports = User;
```

Figura 5. Arquivo user.js, que exporta o *Module* User, entidade que faz a persistência através de outro *module,* o Mongoose.

Parece mais conveniente a descrição do sistema por diagramas relacionados ao comportamento e a macroestrutura de seus componentes, enfatizando a interação entre os módulos do SGRA. Aliás, frequentemente, projetos de sistemas NodeJS são feitos utilizando a técnica de *Behavior-Driven Development* (BDD) [NORTH 2006], que observa a aplicação por seu comportamento em relação aos respectivos "devem-fazer"

das funcionalidades. Essa técnica aplica algumas ideias do *Test-Driven Development* (TDD) pensando nas funcionalidades e no desenvolvimento ágil e no desempenho.

Observa-se que a linguagem Javascript e o NodeJS oferecem recursos e simplificações que, ao mesmo tempo que respeitam princípios de desenvolvimento, também dispensam certos recursos da orientação a objetos em sua forma mais tradicional.

Considerando o escopo deste trabalho, os diagramas de componentes (estrutural) e casos de uso (comportamental) [Guedes 2011] são duas formas, dentre várias, de modelagem do domínio do SGRA.

O diagrama de componentes da figura 6 mostra a estrutura desses dois componentes e suas integrações, da forma como acontece neste sistema e, com algumas modificações pertinentes, poderá representar outros sistemas baseados neste protocolo.

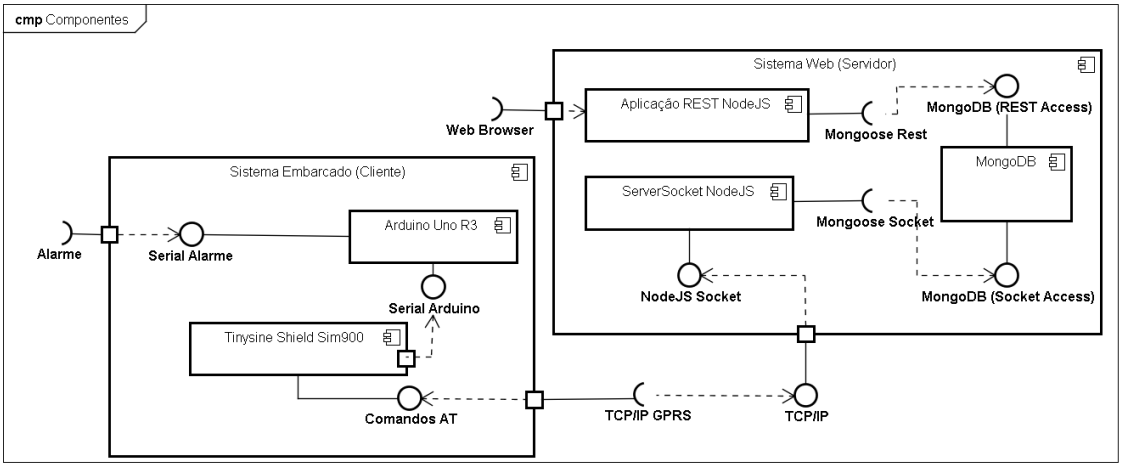

Figura 6. Diagrama de Componentes do SGRA.

No caso do sistema embarcado, as interfaces seriais comunicam Arduino com Sim900 em escrita e leitura. Os comandos AT configuram a conexão com a rede, a transmissão dos dados e a abertura/fechamento do *socket* (TCP). Este sistema se comunica com o sistema web através da rede utilizando um *Access Point Name* (APN) de conexão GPRS em modo cliente. Por sua vez, API em NodeJS atua como servidor de *sockets* TCP recebendo cada requisição e buscando no banco de dados as informações de atualização de status, que são devolvidas ao sistema embarcado. Tudo isso é feito de forma transparente para o usuário, que interage com aplicação web REST amigável utilizando o navegador.

Outra representação UML possível é a de casos de uso. No entanto, o diagrama não pode representar a codificação do sistema embarcado, uma vez que só se aplica a sistemas orientados a objetos [Guedes 2011] e o sistema embarcado é desenvolvido em linguagem procedural derivada da linguagem C [McRoberts 2011]. Como este microssistema desempenha ações no SGRA, ele pode ser visto como um ator do sistema.

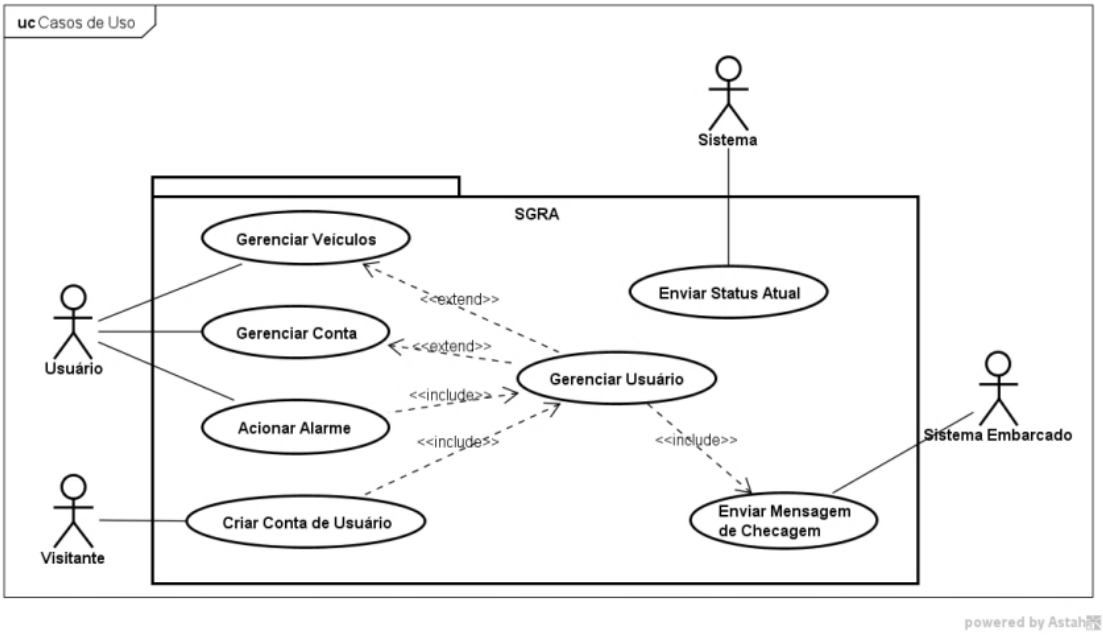

Figura 7. Diagrama de Casos de Uso do SGRA.

Deve-se esclarecer que a autenticação é um pré-requisito para todas os casos de uso do usuário, exceto o caso "criar usuário", e foi retirado para facilitar a leitura do diagrama. Este pré-requisito, de certa forma, é a principal política de segurança do sistema, que poderia ser incrementada ou modificada sem custo de desempenho para o sistema embarcado, o que não aconteceria num sistema de alarme comum. Ou seja, ao invés de uma autenticação com *login* e senha, poderia se utilizar outras formas de autenticação, como leitura biométrica ou autenticação de outros serviços (OAuth2), sem que seja necessária uma modificação no *hardware* do sistema embarcado. Isso diminui o custo na manutenção e na atualização do serviço.

Por sua vez, os casos "gerenciar" são uma simplificação das quatro ações de persistência - CRUD (*Create, Read, Update* e *Delete*), as vezes chamada de "manter". Por fim, a existência de um ator "sistema" apenas indica que o caso de uso correspondente ocorre de forma automática, disparado conforme os próprios critérios da aplicação.

É bom esclarecer que um diagrama de casos de uso deve ser complementado de uma *documentação de casos de uso*, que detalha de forma simples cada caso de uso [Guedes 2011]. No entanto, considerando que esse tipo de documentação possui formato bem flexível [Guedes 2011] e que os procedimentos relacionados a esses casos já foram ou serão detalhados neste trabalho, entende-se que sua ausência neste trabalho não causa danos em relação a documentação de software.

Outro aspecto diferenciado do sistema pode ser observado ao utilizar a base MongoDB, que não é relacional, o que dispensa a utilização de um diagrama entidade-relacionamento [MongoDB 2015]. Aliás, o MongoDB, como já foi dito, pode armazenar diretamente estruturas de atributos variáveis, como objetos e *arrays*. Neste caso, a aplicação foi estruturada para trabalhar com essa flexibilidade.

Uma forma de representação da base MongoDB poderia ser similar à da figura 8.

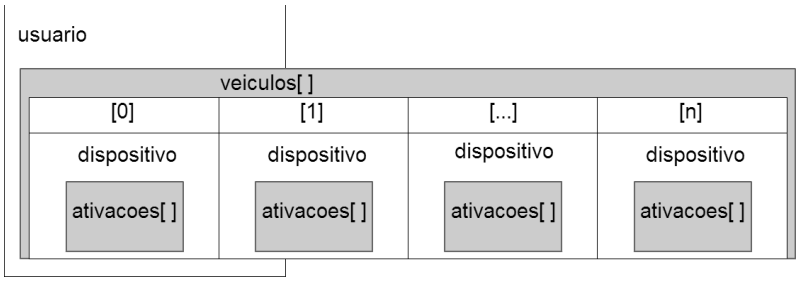

Figura 8. Visualização da *collection* "usuario".

Neste caso, os objetos em cinza são arrays, de tamanhos dinâmicos. Diferentemente de algumas linguagens, o tamanho desse *array* não precisa ser mencionado, o que significa que é possível inserir "ativacoes" e "veiculos" indefinidamente na base de dados. Embora sua estrutura seja dinâmica, vale ressaltar a existência da dependência *mongoose*, que determina um objeto de persistência que, no fim, acaba por delimitar, através de um *schema,* os dados persistentes ao "modelo" da figura. Esse objeto de persistência utiliza o MongoDB Node Driver, a biblioteca padrão do MongoDB para NodeJS. Como o MongoDB também é escrito em Javascript [MongoDB 2015], não há uma conversão dos dados para persistência como ocorre frequentemente em sistemas OO com bancos de dados relacionais.

3.2. Funcionamento Geral do Sistema

Inicialmente, o usuário acessa a aplicação web no endereço IP ou DNS (*Domain Name System*) disponibilizado pela hospedagem, não importando o dispositivo de onde acesse. Lá ele deverá se autenticar, ou criar uma conta para então se autenticar. Em seguida, acessa a área do usuário, onde ele poderá incluir um veículo e associá-lo a um sistema embarcado, como ilustra a figura 9.

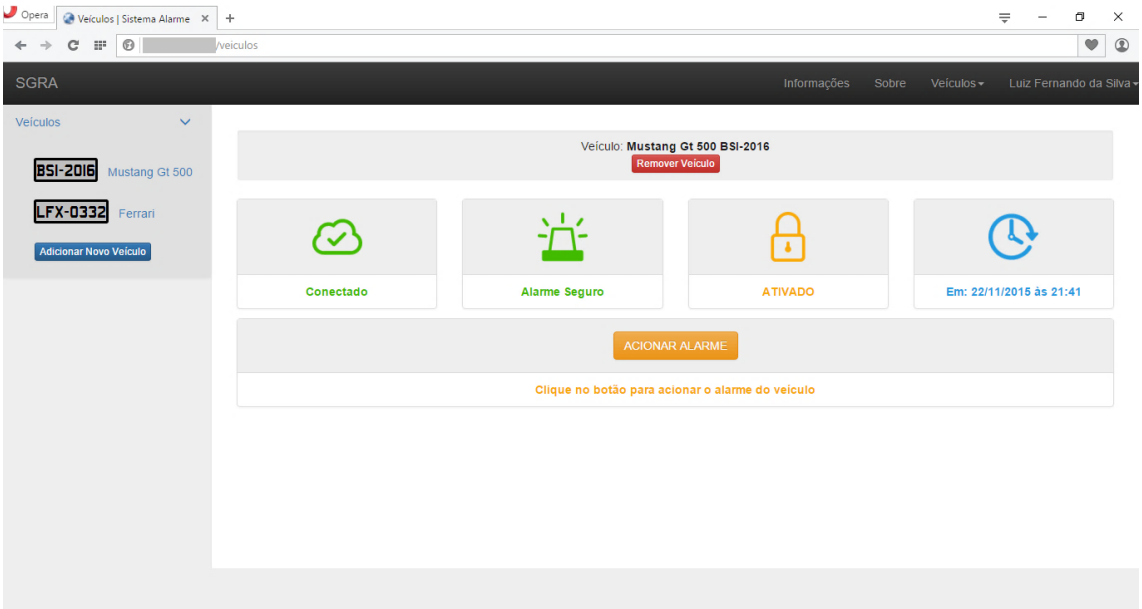

Figura 9. Visualização da "área do veículo".

Como pode ser percebido na figura 9, o usuário visualiza um painel contendo as informações do estado alarme do veículo selecionado, com a possibilidade de acioná-lo individualmente. Para isso, deverá adicionar um novo veículo, informando a placa, a

marca/modelo, a cor e o número serial do sistema embarcado a ser associado, que deverá ser conhecido previamente. Ao fazê-lo, o veículo adicionado se junta aos demais veículos do usuário no painel do canto esquerdo de sua tela, se existirem outros, onde poderá visualizar informações separadas para cada um que selecionar. Cada veículo possui seu próprio cenário de alarme, isto é, o botão para acionamento, data e hora de acionamento, estado atual do travamento e da conexão com o alarme. Deve-se observar também que todas as entradas de dados são validadas em seu formato e tamanho, tanto no navegador quanto no ato de persistência.

Além disso, podem ser visualizadas áreas de instrução de uso e de alteração/exclusão da conta de usuário.

Todas essas operações são apenas de persistência, de forma que não interagem diretamente com o sistema embarcado. Esta é feita através do núcleo ServerSocket.js, uma implementação de servidor *socket* TCP/IP com o módulo "net" do NodeJS, que permite a criação e gerência de *sockets* sem *threads* [Pereira 2013, Casciaro 2014]. O ServerSocket.js fica esperando ("escutando") por conexões, que serão feitas pelo sistema embarcado. Abertos os *sockets*, o sistema embarcado envia algumas informações para o ServerSocket com a finalidade de "saber" o estado atual do alarme no banco de dados, de forma que atualize seu próprio estado caso estes estejam diferentes.

Como foi dito, a representação do alarme foi abstraída para um simples acendimento de *led*: se o alarme está ATIVADO, o *led* deve estar aceso, senão, apagado. A abstração ainda conta com um botão (*interruptor tátil* [McRoberts 2011]), representando um evento qualquer que dispararia um alarme que esteja ativado, como a abertura da porta. Ao ser pressionado o botão e estando o led aceso, o usuário deve visualizar em sua interface a informação de que o alarme está disparando, caso contrário, ser informado de que o veículo está seguro. A figura 10 ilustra a correspondência dessa abstração com o esquema da figura 2.

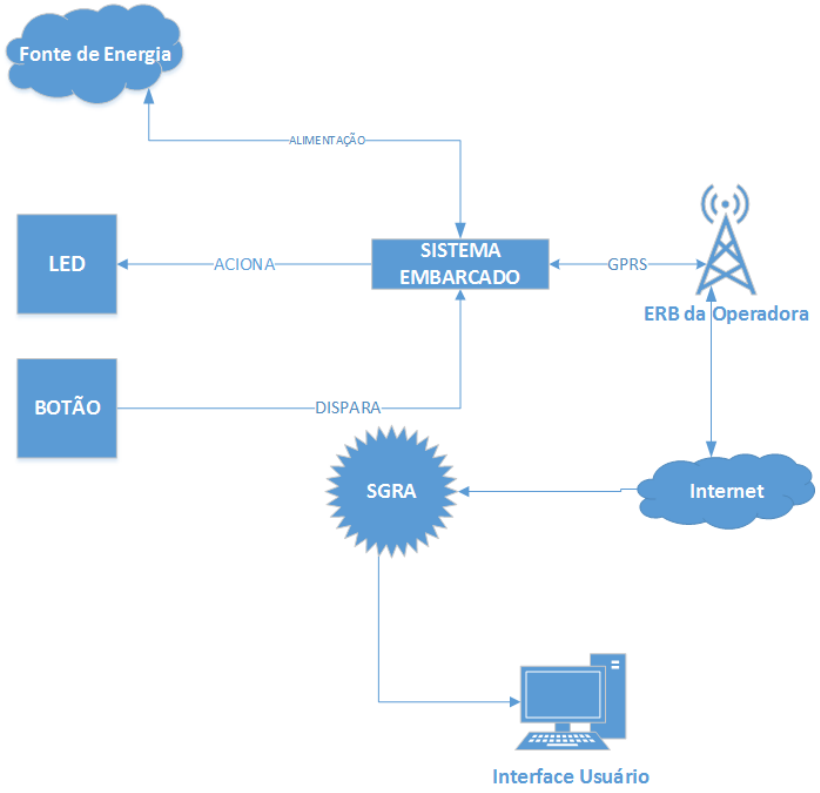

Figura 10. Esquema do alarme abstraído como um *led* e um botão interruptor.

3.3. Comandos AT Diretos contra Biblioteca GSM do Arduino

Do ponto de vista do Arduino, existem duas formas, que não esgotam todas as possibilidades, para o envio de comandos AT pela interface serial: "escrevendo" diretamente, através da biblioteca SoftwareSerial ou utilizando as bibliotecas InetGSM e Sim900.

No primeiro procedimento, configura-se um objeto SoftwareSerial com RX e TX nos pinos 2 e 3, respectivamente, seguindo a documentação da *shield*. A escrita, então, é feita com o método *println* dessa classe. A grande vantagem nessa modalidade está na flexibilidade do uso dos comandos AT, em que é possível configurar o *delay* entre os comandos, por exemplo, conforme a conveniência e a necessidade. Além disso, o método *println* trabalha diretamente com Strings, que são mais simples de manipular, concatenar e dividir do que *arrays* de caracteres, já que oferecem funções que simplificam isso. Ainda é possível personalizar os tempos de *delay*.

No segundo procedimento, utiliza-se a abstração obtida com as bibliotecas Sim900 e InetGSM, através de um objeto SIMCOMM900 (da biblioteca do SIM900) em conjunto com um objeto InetGSM. Enquanto o primeiro faz a comunicação serial, o segundo realiza funções chave como checagens da rede GSM, conexão com APN e abertura do *socket*. Este procedimento tem a vantagem de usar os mesmos comandos AT de forma transparente, o que proporciona uma conexão e uma troca de dados mais estáveis. Essas bibliotecas, no entanto, apresentam um relativo inconveniente de só trabalharem com *arrays* de caracteres e ponteiros para eles, que são Strings na forma tradicional da linguagem C, o que aumenta um pouco o trabalho de programação. Além disso, não se tem o controle direto sobre os *delays*.

3.4. Tráfego de Dados

O sistema embarcado apenas envia e recebe dados, deixando o processamento pesado para a aplicação web, o que é interessante no ponto de vista de se estabelecer políticas mais elaboradas de segurança. Sendo assim, as funções do sistema embarcado são: enviar sua identificação, receber e tratar a resposta do servidor, atualizar seu status (acendendo o led adequado), enviar confirmação ao servidor, aguardar por eventos de alarme disparado (pressionamento do botão) e enviar notificação ao servidor.

4. Análises e Resultados

Com o protótipo em funcionamento, é necessário levantar algumas questões relativas ao contexto, técnicas, falhas e possibilidades. A figura 10 mostra o sistema embarcado construído em funcionamento, utilizando uma *protoboard* para realizar todas as ligações elétricas requeridas.

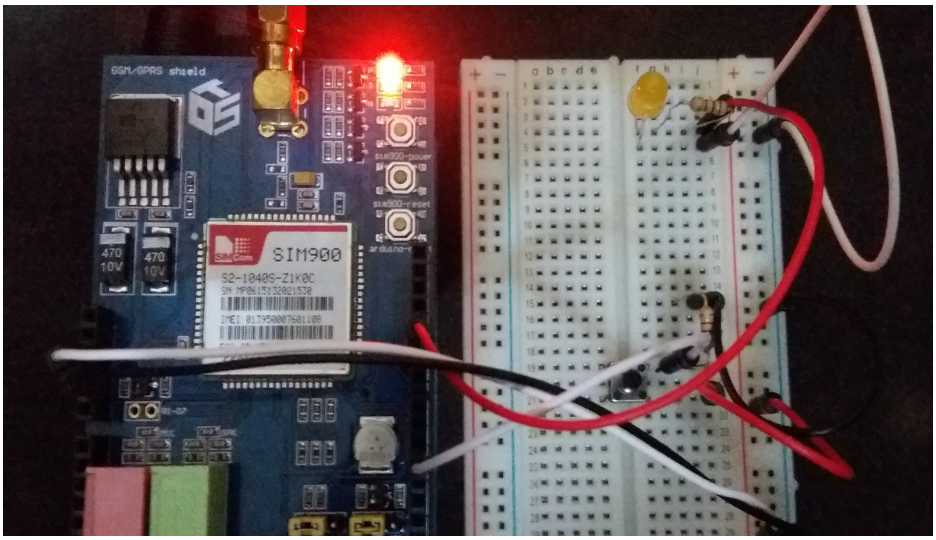

Figura 11. Sistema Embarcado do Protótipo em Funcionamento.

4.1. Procedimento de Escrita Adotado

O procedimento de escrita na porta serial está intimamente ligado ao sucesso de todo o sistema, uma vez que determina não só o acionamento e a gerência dos eventos, mas a velocidade com que tudo isso ocorre. Estes foram descritos no capítulo 3.3.

O primeiro procedimento apresentou relativa instabilidade, com fechamentos inesperados dos *sockets* e/ou não recebimento das mensagens do servidor, além de *timeouts* aparentemente sem explicação e demora nos casos de conexão bem-sucedida. Esses fatores podem estar relacionados aos tempos de *delay* inadequados e omissão de comandos AT.

Já o segundo procedimento atendeu bem às expectativas, em que o processo ocorre mais rápido e de forma mais estável. No entanto, foi necessária uma pequena modificação na biblioteca InetGSM, diminuindo alguns *delays*, quando se obteve o melhor tempo para inicialização (cerca de 1 minuto) e de atualização de status (entre 1 e 3 segundos). Com isso, os acionamentos ficaram visivelmente mais rápidos.

Por sua vez, as notificações de “alarme seguro” e “alarme disparando” são renderizadas no *browser* através de requisições AJAX com a biblioteca JQuery, num

intervalo de 500ms. Este valor se soma ao ciclo de 1s da frequência de envio do estado do Arduino ao servidor.

Com a escolha do método de escrita, realizaram-se testes em três regiões de Juiz de Fora, nos três diferentes turnos (manhã, tarde e noite), cujos resultados estão expostos na Tabela 1. Nestes casos, foram comparados os tempos de estabelecimento de conexão ("inicialização"), da primeira troca de mensagens entre sistema embarcado e servidor ("primeira atualização"), de acionamento remoto do alarme via navegador ("acionamento") e o total entre pressionamento do botão na plataforma Arduino e renderização no navegador ("notificação"). Os tempos foram muito similares, o que era esperado, dado que as três regiões possuem cobertura GSM 2G ou superior. Com esse teste, pode-se atestar que o funcionamento do sistema é consistente e não depende, por exemplo, do tráfego de rede em determinado horário. Como já foi dito, fatores ambientais podem explicar as variações. Destaca-se também que essas medições consideram que o terminal de acionamento está em conexão estável com a Internet.

Tabela 1. Medições (em segundos) de inicialização, primeira atualização, acionamento e notificação.

	Região 1			Região 2			Região 3			
Métrica	Manhã	Tarde	Noite	Manhã	Tarde	Noite	Manhã	Tarde	Noite	Média
Inicialização	57	58	53	49	50	49	55	57	54	53,5
Primeira Atualização	23	23	23	21	21	23	23	24	23	22,6
Acionamento	3	3	3	3	4	2	3	4	2	3
Notificação	3	3	3	3	3	3	3	4	3	3,1

4.2. Considerações sobre o Sistema Embarcado

Embora o sistema embarcado deste protótipo seja composto por circuitos genéricos e ele execute satisfatoriamente as operações do sistema desta forma, nada impede a construção de um projeto de circuito que atenda mais especificamente aos requisitos e a novas funcionalidades. Esta consideração vai de encontro à possibilidade de expansão, que deve também ser considerada no caso do surgimento de novos requisitos. Assim, é necessário um estudo minucioso dos requisitos e do domínio do sistema.

4.3. Avaliação Quanto ao Problema dos Alarmes Comuns

Considerando o problema descrito, pode-se dizer que o sistema não necessariamente o resolve, já que os roubos de carros não são problemas intrínsecos da tecnologia, mas socioeconômicos e políticos. No entanto, ele cria uma dificuldade para esse tipo de crime.

Diferentemente do sistema de alarme UHF, a rede de telefonia celular oferece maior segurança no tráfego de dados, o que inviabiliza uma possível cópia da chave. No entanto, ainda é possível bloquear o recebimento do comando através de um bloqueador de celular, que pode ser comprado, mas é mais caro. Lembrando da possibilidade dos ataques "*man-in-the-middle*", que necessitam de condições bastante específicas. Além disso, a existência de um sistema web com políticas de segurança bem consolidadas inviabiliza outras formas de acionamento e tentativas de invasão.

Além disso, já existem dispositivos capazes de anular a atuação dos "chapolins", conhecidos como *anti-jammers*, que, no entanto, não acionam o alarme.

Do ponto de vista regulatório, pode-se dizer que o sistema permanece atendendo às principais normas estabelecidas, seja no âmbito do serviço, seja no âmbito das telecomunicações, pois utiliza serviços que já existem e são regulamentos.

Logo, pode-se dizer que o sistema protótipo é uma alternativa funcional e segura de interação *real time* com o veículo.

5. Considerações Finais

Este trabalho propôs um projeto e um protótipo de sistema que busca a gerência remota de alarmes veiculares e suas especificações, bem como seu suporte teórico. Além disso, propôs um protocolo de acionamento funcional que pode ser utilizado em vários outros contextos de automação, telemetria e outros procedimentos remotos.

5.1. A Informática no Contexto IoT

Um trabalho no âmbito de *Internet of Things* expõe facilmente a interdisciplinaridade entre as áreas de computação e engenharias de automação (mecatrônica), que deve ocorrer com mais frequência nos próximos anos. Neste contexto, sistemas computacionais não mais se resumem a softwares para usuários, mas sistemas que interagem com máquinas e outros sistemas de grande criticidade.

Destarte, este trabalho também leva a uma reflexão sobre o modelo atual de ensino de Computação e Engenharias e quão prontas estão os egressos a atuarem nessas áreas diante dos desafios decorrentes da Internet das Coisas e dos sistemas integrados de automação.

Também não se deve deixar de mencionar o papel da Internet conectando objetos a sistemas sofisticados e retirando informações de entidades que antes não eram monitoráveis, especialmente em veículos e objetos móveis. Tais objetos podem se tornar cada vez mais inteligentes e completos, podendo assim, realizar cada vez mais tarefas.

5.2. Trabalhos Futuros

A expectativa é que este trabalho seja estendido e incorpore outras formas de medição e controle, atuando em outros segmentos do veículo ou mesmo de outros equipamentos eletrônicos. Dado o escopo trabalhado por esta aplicação, várias são as oportunidades de melhoria e de correções, dentre as quais algumas merecem destaque.

O primeiro ponto é em relação ao uso da rede de telefonia. É necessário trabalhar em procedimentos alternativos de acionamento para casos em que a mesma esteja indisponível ou não ofereça as condições de conectividade 2G/3G adequadas, para que, nessas hipóteses, o usuário possa ao menos utilizar o alarme da forma tradicional, sem a utilização remota. Algumas alternativas, aplicadas adequadamente ao caso, podem ser *bluetooth,* infravermelho, conexão em modo *access point,* etc.

Além disso, dadas as fragilidades de segurança já citadas da rede de telefonia, se faz necessária a adoção de políticas auxiliares de segurança e criptografia. Essas políticas de segurança, aliás, devem contemplar não só a conectividade com a rede, mas

também ao próprio servidor de aplicação, como o uso de *firewalls*. Também é interessante a utilização das redes 3G e 4G quando estiverem disponíveis, o que vai necessitar de um módulo GSM mais sofisticado, mas que permitirá a utilização de outros serviços que forem convenientes. Aliás, dado que as políticas de segurança estarão implementadas em nuvem e não no próprio dispositivo embarcado, a tendência é que a segurança do sistema naturalmente seja associada à segurança da rede dos dispositivos.

Outro ponto é a política de autenticação, que pode admitir uma ou várias formas para autenticação, como a autenticação externa (OAuth2), biométrica, e/ou vinculada ao dispositivo, entre outras.

Parece bastante promissor também a possibilidade de o sistema embarcado tratar outras informações do veículo como nível de combustível, velocímetro e coordenadas geográficas, caso tenha um GPS, e outros dados que sejam processáveis por sistemas de análise inteligente. Também é interessante a possibilidade de se enviar outros comandos ao veículo. Em linhas gerais, o protocolo aplicado no SGRA é relativamente genérico, o que implica que é possível trabalhar com outras medições em outros contextos, como coordenadas de GPS, níveis de fenômenos climáticos, fotoelétricos, pressão, temperatura, umidade, entre outros, considerando um sistema embarcado adequado.

Também pode ser interessante a criação de um alarme automotivo que já suporte nativamente essas funcionalidades, oferecendo o serviço adicionalmente, por exemplo.

Por fim, não pode ser desconsiderado o fato de existirem protocolos próprios para a plataforma IoT [Martins e Zem 2015] que podem ser comparados sob determinadas condições.

Referências Bibliográficas

Almeida, Letícia (2015) "Os 50 Carros Mais Roubados no Brasil", http://exame.abril.com.br/seu-dinheiro/noticias/os-50-carros-mais-roubados-do-brasil. Acesso em 23 Jan 2016.

Agência Nacional de Telecomunicações (ANATEL) (2008) "Resolução nº506 de 01/07/2008".

Akerkar, Rajendra (2014) "Big Data Computing", Songdal, CRC Press.

Beighley, Lynn (2007) "Head First: SQL", Sebastopol, O'Reilly.

Casciaro, Mario (2014) "Node.js Design Patterns", Birgminham, Packt.

Dayley, Brad (2014) "Node.js, MongoDB and AngularJS Web Development", Michigan, Pearson Education Inc.

Departamento Nacional de Trânsito (DENATRAN) (2014) "Frota", http://www.denatran.gov.br/frota.htm. Acesso 05 Fev 2015.

Elmasri, Ramez; Navathe, Shamkant B. (2005) "Sistema de Banco de Dados", São Paulo, Pearson Addison Wesley.

Guedes, Guilherme T.A. (2011) "UML2 – Uma Abordagem Prática", São Paulo, Novatec.

Gomes, Luis Flávio (2015) “Roubos e Furtos de Veículos no Brasil”, Editorial do Instituto Avante Brasil, http://institutoavantebrasil.com.br/roubos-e-furtos-de-veiculos-no-brasil/. Acesso 23 Jan 2016.

Lima, Adilson da Silva (2012) “Especificações Técnicas de Software”, São Paulo, Erica.

Luz, Marlon; Humenhuk, Lucas (2015) “Microsoft Virtual Academy: A Internet das Coisas – Fundamentos de IoT”, https://www.microsoftvirtualacademy.com/pt-br/training-courses/a-Internet-das-coisas-fundamentos-de-iot-12622. Acesso em 21 Nov 2015.

Mardan, Azat (2014) “Practical Node.JS: Building Real-World Scalable Web Apps”, Nova York, Apress.

Martins, Ismael Rodrigues; Zem, José Luiz (2015) “Estudo dos Protocolos de Comunicação MQTT e COAP Para Aplicações Machine-to-machine e Internet das Coisas”, Revista Tecnológica da FATEC Americana, v.3, n.1, p. 64-87.

McRoberts, Michael (2011) “Arduino Básico”, 1ª Ed. São Paulo, Novatec.

Margolis, Michael (2011) “Arduino Cookbook”, Sebastopol, O’Reilly.

Molisch, Andreas F (2011) “Wireless Communications”. 2ª Ed. Nova York, Wiley.

Medeiros, Júlio Cesar de Oliveira (2007) “Princípios de Telecomunicações: Teoria e Prática”, 2ª Ed. São Paulo, Erica.

MongoDB University (2015) “M101JS: MongoDB For Node Developers”, https://university.mongodb.com/courses/M101JS/about. Acesso em 10 Mai 2015.

Moroz, Maiko R. et al (2012) “Requisitos para a adoção de Sistemas Operacionais Embarcados”, Revista Visão Eletrônica, ano 5, n.2, pp. 90 a 97.

Nascimento, Rodrigo (2015) “O que, de fato, é Internet das coisas e que revolução ela pode trazer?”, http://computerworld.com.br/negocios/2015/03/12/o-que-de-fato-e-Internet-das-coisas-e-que-revolucao-ela-pode-trazer. Acesso em 10 Out 2015.

North, Dan (2006) “Introducing BDD”, http://dannorth.net/introducing-bdd/". Acesso em 15 Nov 2015.

Pereira, Caio Ribeiro (2013) “Aplicações web real-time com NodeJS”. São Paulo, Casa do Código.

Saudate, Alexandre (2013) “REST - Construa API's inteligentes de maneira simples”, São Paulo, Casa do Código.

Schmidt, Douglas, et al (2000) “Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2”, 2ª Ed. Nova York, Wiley.

Silva, Alberto Manuel Rodrigues; Videira, Carlos Alberto Escaleira (2001) “UML, Metodologias e Ferramentas CASE”, Lisboa, Centro Atlântico.

Sommerville, Ian (2003) “Engenharia de Software”, São Paulo, Addison Wesley.

Stefanov, Stoyan (2010) “JavaScript Patterns”, Sebastopol, O’Reilly.

Tanembaum, Andrew S (2009), “Sistemas Operacionais Modernos”, São Paulo, Pearson.

Zakas, Nicholas C (2014) “The Principles of Object-Oriented JavaScript”, San Francisco, No Starch Press.