PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

Monitoramento e Gerenciamento Remoto de Nós Móveis em Campo Demarcado por Áreas

Davi Rocha Silva

PROJETO FINAL DE GRADUAÇÃO

CENTRO TÉCNICO CIENTÍFICO - CTC

DEPARTAMENTO DE INFORMÁTICA

Curso de Graduação em Ciência da Computação

Rio de Janeiro, Janeiro de 2014

Davi Rocha Silva

Monitoramento e Gerenciamento Remoto de Nós Móveis em Campo Demarcado por Áreas

Relatório de Projeto Final, apresentado como requisito parcial para a obtenção do titulo de Bacharel em Ciência da Computação.

Orientador: Markus Endler

Rio de Janeiro

Janeiro de 2014.

Resumo

Rocha Silva, Davi. Endler, Markus. Monitoramento e Gerenciamento Remoto de Nós Móveis em Campo Demarcado por Áreas. Rio de Janeiro, 2013. 364p. Relatório Final de Projeto Final II – Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro.

Esse trabalho descreve o desenvolvimento de um sistema de monitoramento e gerenciamento remoto de nós móveis executando a plataforma Android. Usando GPS e redes móveis 2G/3G, esses nós são monitorados em tempo real através de uma central de controle.

Palavras-chave

DDS, SDDL, Android, Gerenciamento, Internet, Web

Abstract

Rocha Silva, Davi. Endler, Markus. Remote Monitoring and Management of Mobile Nodes in the Field with demarcated Geographic Areas. Rio de Janeiro, 2013. 364p. Final Report for Final Project II – Department of Informatics. Pontifical Catholic University of Rio de Janeiro.

The proposal of this Project is to develop a system that will monitor and control the mobile nodes, this nodes will be monitored on real time using a GPS and a control panel.

Keywords

DDS, SDDL, Android, Management, Internet, Web

Sumário

Lista de Figura

1. Introdução

1.1 Motivação

Um crescente número de atividades econômicas envolvem a operação de equipamentos móveis, frotas de veículos ou funcionários em campo que precisam ser gerenciados e monitorados remotamente através de uma central de controle. Dentre as inúmeras aplicações com equipamentos móveis que requerem um monitoramento e gerenciamento remoto, o projeto Motofog[1] da empresa Fumajet foi escolhido pelo fato de estar ligado diretamente a projetos sociais, ajudando no combate da dengue e outras pragas. O Motofog é um dispositivo que é transportado em motocicletas para fazer um fumacê (com o produto a ser aplicado) utilizando o escapamento da motocicleta. O Motofog tem a vantagem de conseguir chegar a locais de difícil acesso, como ruelas e becos estreitos em comunidades carentes, o que não é possível com veículos maiores, normalmente usados nessas atividades.

1.2 Definição do problema

Com o produto Motofog desenvolvido e sendo comercializado no mercado~~Apesar de terem desenvolvido o Motofog~~, faltava à Fumajet um sistema para o rastreamento, monitoramento e gerenciamento que permitisse um acompanhamento ~~e gerenciamento remoto da operação~~ de seus equipamentos em campo em tempo real e off-line, e que esteja integrado com o sistema Motofog. Sem tal acompanhamento é impossível saber em quais áreas o produto (de combate à praga) foi aplicado, se o produto foi aplicado na área correta, e se o operador da moto está utilizando a vazão correta na hora da aplicação. Então, visando suprir essa necessidade da empresa, surgiu a oportunidade de desenvolver um sistema de rastreamento e gerenciamento remoto para o equipamento Motofog. Esse sistema terá as seguintes características: será capaz de mostrar para uma central de controle (uma aplicação Web) a localização das motos em um mapa em tempo real, será capaz de informar o nível de vazão da bomba do Motofog quando o operador estiver aplicando o produto, permitirá definir áreas geográficas com a função de “cerca eletrônica”, para saber se o operador da moto está aplicando o produto nas áreas pré-determinadas, e possibilitará a geração de relatórios sobre o estado de operação de cada equipamento Motofog ao longo do tempo, para futuras análises e controle, possibilitando assim fazer um melhor planejamento da aplicação do fumacê para combate às pragas.

O sistema de monitoramento e gerenciamento remoto também inclui um dispositivo de hardware acoplado ao Motofog que utiliza redes de telefonia móvel para enviar para a central de controle a posição geográfica da moto, o nível de vazão da bomba no momento da aplicação do produto, e a informação se a bomba está ligada ou desligada. O dispositivo de hardware para o Motofog foi desenvolvido pela empresa ER2 utilizando a placa IOIO-OTG [16]. Esta placa integradora provê uma interface com hardware acoplado, no caso a bomba do Motofog, que pode ser acessada utilizando a plataforma Android. Assim, faz parte desse projeto final também o software do cliente mobile utilizado para interação com o IOIO-OTG e envio dos dados para a central de controle.

1.3 Justificativa da relevância do problema e da ausência de soluções satisfatórias

O problema de monitoramento e gerenciamento dos equipamentos Motofog é muito específico e demanda um sistema customizado para atender às necessidades específicas da empresa, tais como visualizar o nível de vazão da bomba e/ou o controle da aplicação do produto em áreas geográficas pré-definidas. Outros requisitos incluem um baixo custo de operação e de manutenção do sistema.

Uma~~ pesquisa~~ estudo de mercado das principais empresas de rastreamento revelou que os seus produtos só permitem o rastreamento simples de veículos (e um bloqueio remoto dos mesmos), sem permitir que se acople os rastreadores a equipamentos específicos, como por exemplo o Motofog, para monitorar o seu estado de funcionamento. Além disso, algum ~~a maioria~~ dos produtos ~~também~~ não possibilitam a visualização on-line dos veículos em um mapa, e a definição de áreas geográficas (cercas eletrônicas).

Outro fator limitante é a forma de comunicação desses dispositivos, que geralmente utilizam protocolos de mercado que são orientados a conexão e apresentam baixo desempenho em situações de conectividade intermitente, além de muitas vezes apresentarem~~m~~ problemas de escalabilidade com relação ao número de conexões simultâneas com a central de controle. Assim, para possibilitar uma comunicação confiável, robusta a desconexões temporárias, que utilize poucos recursos de rede e do dispositivo móvel, e que seja escalável, será utilizado o middleware SDDL[2], desenvolvido no laboratório LAC da PUC-Rio. O SDDL possuí um protocolo de comunicação mobile, o MR-UDP[20], que sendo baseado em UDP, é bem leve, escalável e eficiente na presença de desconexões temporárias.

2. Estado da Arte

Soluções de hardware para a empresa teriam que permitir o monitoramento tanto da localização como também do estado de operação do equipamento Motofog. No mercado atual existem várias soluções de rastreamento para veículos, inclusive motocicletas, mas nenhuma apresenta a possibilidade de customização para satisfazer as necessidades da empresa. Existe apenas a possibilidade de acoplar o dispositivo de rastreamento ao painel da moto para obter informações sobre posição, velocidade, torque, frenagem da moto, e algumas outras informações sobre o funcionamento do motor e demais dispositivos da moto. Algumas empresas oferecem soluções para o rastreamento de dispositivos específicos, mas os altos custos de customização, configuração e manutenção geralmente tornam o desenvolvimento e uso inviável para pequenas empresas, ou empresas que tenham um número médio (centenas ou dezenas de milhares) de equipamentos/veículos a serem monitorados.

Quanto ao software para rastreamento de veículos existem vários produtos interessantes e com funcionalidades bastante sofisticadas. Os sistemas analisados geralmente possuem a funcionalidade básica de mostrar em um mapa o veículo (a moto) em sua localização corrente, enquanto que serviços como cercas eletrônicas geralmente são vendidos à parte. Porém, todos os softwares não são capazes para suprir as necessidades específicas da empresa em questão. Por exemplo, nenhum permite o monitoramento customizado de equipamentos específicos, i.e. saber se a bomba do Motofog está ligada ou

desligada, e qual é o seu nível de vazão do produto durante a aplicação do mesmo.

Portanto, dada a ausência de produtos de hardware e software no mercado que pudessem ser usados diretamente para o monitoramento dos equipamentos Motofog, a solução proposta foi desenvolver uma solução personalizada incluindo um hardware especifico utilizando o IOIO-OTG para interagir com o Motofog, o software correspondente para o lado cliente (uma aplicativo Android), e um serviço web para a central de controle customizada com todas telas e as funcionalidades necessárias. E para a comunicação entre o cliente móvel e a central de controle optou-se pelo uso do middleware SDDL, devido ao seu suporte para comunicação wireless e com um grande número de clientes mobile. A solução personalizada também possui a vantagem de facilitar a manutenção do sistema, permitindo que futuras adaptações e alterações no sistema possam ser feitas a um custo menor, e de forma mais ágil.

3. Proposta

Antes de ser fechada a proposta do projeto, foi apresentado ~~a~~à Fumajet um vídeo demonstrando o funcionamento da "cerca eletrônica" e o comportamento dos ícones no mapa. Se a moto estivesse fora da "cerca eletrônica" o ícone ficaria de uma cor. Caso a moto estivesse dentro da área com a bomba ligada, teria outra cor e assim por diante.

Com o vídeo foi possível esboçar um rascunho rápido da idéia, para ter o aceite ou não da empresa e ver se essa era uma solução que iria suprir as suas necessidades. A Fumajet aprovou o vídeo e logo em seguida deu-se o início a criação de uma proposta mais detalhada do que precisava ser feito no projeto.

A decisão da construção do hardware e do estudo do sistema Motofog para a integração ficou a cargo do aluno Marlon Moura (aluno de E~~e~~ngenharia de C~~c~~ontrole e A~~a~~utomação da PUC-Rio).

Com o excelente trabalho do aluno Marlon Moura concluído e já com a integração do hardware com o sistema Motofog.,~~Uma vez que dispositivo de hardware para interação com o Motofog estava construído,~~ o desenvolvimento do software do sistema completo envolveu as seguintes tarefas:

1) Estudo da biblioteca do IOIO, para entender como o Android obtém os valores dos sinais da placa e passa para o sistema.
2) Implementação do cliente móvel para a plataforma Android, que fosse capaz de: interagir com a placa integradora (para captar os sinais de controle da bomba Motofog), obter a localização corrente do dispositivo e utilizar a ClientLib do SDDL para enviar essas informações para a central de controle.
3) Desenvolvimento de uma aplicação Web para a central de controle acessível, com as respectivas funcionalidades de visualização e controle necessárias para o gerenciamento de uma frota de Motofogs.
4) Utilização e configuração do middleware SDDL para a comunicação entre o nó móvel e a central. O núcleo do SDDL foi implantado em um serviço de nu~~ú~~vem.
5) Criação de um simulador junto ao aplicativo que roda no dispositivo capaz de simular várias motos que enviam coordenadas GPS e estados da bomba. O simulador possui botões para a interação e alterar os estados das motos.

Central de controle web

A central será capaz de mostrar as motos em tempo real utilizando um mapa, terá controle de áreas que servirão como cercas virtuais, irá fazer o controle de cada unidade que terá um identificador associado a cada Motofog, terá controle de acesso de usuários e irá gerar relatórios aonde terá o total de quilometragem ~~a quantidade de quilometragem~~ que cada moto andou e em quais áreas ela passou.

Middleware SDDL

O middleware é responsável pela a comunicação do cliente móvel com a central de controle, para envio da posição corrente da motocicleta e de várias informações obtidas da interface com o Motofog. Essa comunicação cliente-central ocorre através de uma entidade do SDDL chamada gateway que executa o protocolo MR-UDP. O gateway executa em uma máquina estacionária e com IP público, e gerencia a conexão com muitos clientes simultaneamente.

O Cliente Móvel para Android

O cliente móvel executará em um smartphone ~~convencional~~(o smartphone precisa possuir acesso 2G/3G, chip GPS, Bluetooth e versão Android 2.3 ou superior), utilizará a biblioteca ClientLib do middleware SDDL e será composto de vários componentes **Service** do framework Android executando em threads separadas, e interagindo entre sí através de **Broadcasts** locais. Um Service Android é um componente que executa em background, e não possui interface com o usuário, já **Broadcasts** são mensagens assíncronas difundidas para todoas oas componentes que fazem parte de uma aplicação Android, e capturadas através de **Broadcast Receivers**.

Visão geral do dispositivo para Interação com o Motofog

O hardware é uma componente importante do sistema. Inicialmente consideramos a possibilidade de projetar placas específicas “do zero”, para fazer~~fazendo~~ a integração com o sistema Android, mas nos estudos sobre os hardwares disponíveis~~l~~ no mercado e sobre os componentes, achamos o IOIO-OTG[16]. Trata-se de uma placa open source (vide Figura 1) criada pelo Google para integração com o sistema Android e que possui já uma biblioteca pronta para utilização.

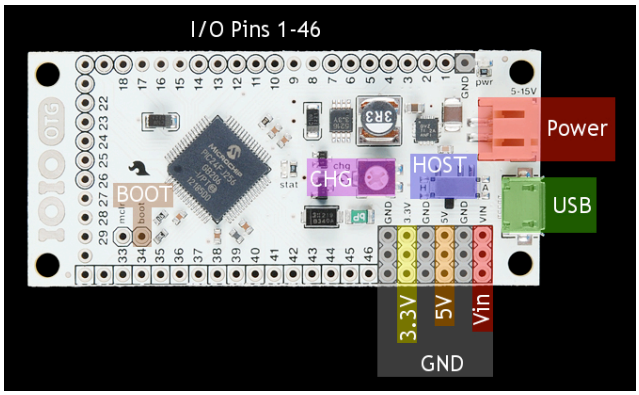

Figura 1: Placa IOIO-OTG

Conforme mostrado na Figura 2, o dispositivo desenvolvido para a integração com o sistema Motofog é composto de um smartphone Android ~~convencional~~ conectado via cabo mini-USB à placa IOIO-OTG. O IOIO-OTG, por sua vez, está soldado a uma placa instrumentadora que está ligada aos cabos de controle da bomba do Motofog. Assim, é possível medir no smartphone (através da interface de controle do IOIO-OTG) os sinais elétricos que passam nos cabos da bomba. Isso permite verificar se a mesma está ligada ou desligada e qual é o nível de vazão da bomba do Motofog, que pode ser de 1 a 4. Através do IOIO-OTG e sua placa integradora o smartphone também obtém a sua fonte de energia.

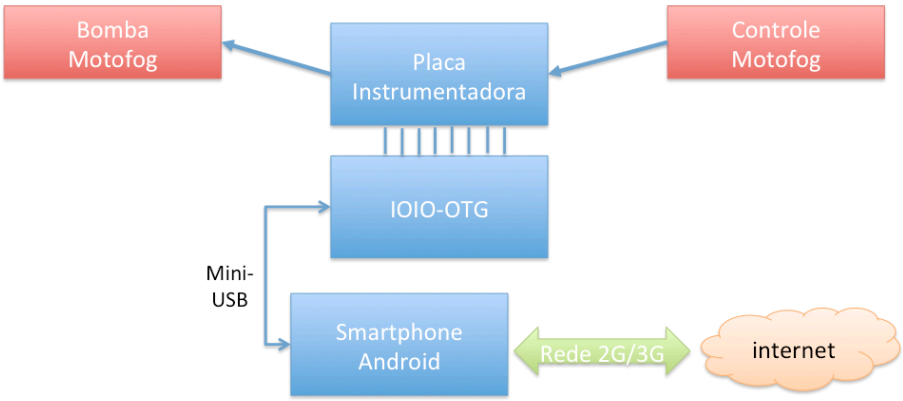

Figura 2: Dispositivo para Interface com o sistema Motofog

A Figura 2 mostra que o IOIO-OTG e a placa instrumentadora interceptam os sinais vindos do controle para a bomba Motofog e disponibilizam esses sinais para o Smartphone, que pode acessá-los através da API correspondente no Android. O Smartphone então usa a sua conectividade 2G/3G para enviar as informações através da internet, para a central de controle.

4. Atividades realizadas

4.1 Estudos e conceituais de tecnologia

Central de controle – Linguagem e Framework

Foi decidido que a central de controle seria desenvolvida como uma aplicação Web, para que o monitoramento pudesse ser feito através de qualquer browser web. Para tal fez-se necessária a avaliação de tecnologias web e uma escolha de qual seria a melhor opção a ser utilizada.

A primeira parte foi a escolha da linguagem de programação. As principais linguagens sendo usadas atualmente para programação web são: PHP, Python, Ruby e Java. No caso, todas as linguagens apresentam funcionalidades e facilidades similares~~, aparecendo apenas como destaque a linguagem Java, que utiliza Servlet~~. A linguagem específica Ruby utiliza um framework chamado Ruby on Rails para a criação de websites. O Python também é uma opção interessante, já que disponibiliza vários frameworks que auxiliam na criação de websites. Finalmente, o PHP, que acabou sendo a linguagem escolhida por ser uma linguagem bastante usada no mercado e com muitos desenvolvedores. Isso facilitará encontrar programadores para futuramente dar suporte e continuidade ao projeto, caso isso se faça necessário.

A segunda parte foi escolher uma plataforma de desenvolvimento, já com a linguagem PHP definida. Foram então realizados estudos e testes de alguns frameworks em PHP bastante utilizados no mercado.

O primeiro, Zend Framework[3], é um framework muito extenso e com muitos módulos, mas ele não foi escolhido por demandar um estudo muito extenso do seu funcionamento. P~~; já p~~ara tarefas simples,~~.~~ p~~P~~or exemplo, na parte de implementação de autorização e autenticação precisa-se lidar com várias interfaces dos componentes e rotas de acesso, tornando o processo bem complexo e demorado. Além disso, a documentação do framework é pouco detalhada e incompleta, dificultando ainda mais o processo de aprendizado.

Como segunda opção considerou-se o CodeIgniter[4], mas que foi logo descartada porque o CodeIgniter não vem acompanhando a rápida evolução da linguagem PHP. Por exemplo, o CodeIgniter utiliza a versão PHP 5.2 e a atual versão estável do PHP é a 5.4. Outro motivo para descartar essa opção é a incerteza sobre a continuação do projeto CodeIgniter, já que a empresa EllisLab[5] está procurando por outro patrono para o projeto.

Já o framework CakePHP[6] mostrou ser muito lento. Um ponto de bastante preocupação é que o código do framework é confuso e não muito bem estruturado. Alguns componentes tem comportamentos não desejáveis, gerando anomalias estranhas, então foi descartado.

O quarto framework avaliado foi o Laravel[7]. Esse framework é de fácil instalação e configuração, pois possui uma ferramenta de instalação e atualização automatizada, e a documentação é simples e bem objetiva, as vezes até simples de mais com exemplos triviais, deixando a desejar em alguns pontos. O Lavarel é compatível com as ultimas versões de PHP e os códigos internos do framework são muito bem estruturados e bem comentados, deixando claro o que cada método e componente faz. Assim, localizar uma classe para estudar o código e entende-la é extremamente fácil e rápido. O framework é bem rápido e já provê vários recursos para se criar uma API de acesso ao sistema. A principal vantagem do Laravel é permitir uma programação enxuta da aplicação e de forma bem elegante. Já nos primeiros testes foi possível perceber que com poucas linhas de código pode-se implementar componentes totalmente funcionais, e tendo a opção de se utilizar muit~~o~~as outr~~o~~as componentes já existentes do Laravel.

Então, por ser de fácil utilização e rápido desenvolvimento a escolha final foi o Laravel para a construção da parte web.

Central de controle – Banco de dados

Dois bancos de dados foram avaliados para esse projeto, o MySQL[8] e o PostgreSQL[9].

O MySQL é um banco de dados bastante difundido e bastante eficiente. Mas depois de um estudo mais detalhado, algumas deficiências (com relação ao tratamento de dados) foram notadas. A seguir, seguem alguns exemplos:

1 – Em uma coluna de texto com *NOT NULL*, quando uma linha é inserida o MySQL não gera nenhum alerta e simplesmente insere uma string em branco. Nesse caso o mais adequado seria alertar o usuário sobre a tentativa de inserir uma linha com um valor nulo.

2 – Em coluna *Timestamp* (para armazenar data e hora) também com a restrição de *NOT NULL*, quando se insere a linha uma nova linha o MySQL coloca valor totalmente sem sentido como *0000-00-00 00:00:00*, que seria o ANO-MES-DIA HORA:MINUTO:SEGUNDO tudo em branco.

3 – Em uma coluna com a declaração de *NUMERIC(2,0)*, espera-se um número com duas casas da parte inteira, por exemplo 90 ou 11. Inserindo o número 100, o MySQL simplesmente trunca o número para 99, alterando assim o valor original. O correto seria o MySQL informar que o número sendo inserido possui três partes inteiras e não duas.

O principal problema é que o MySQL assume valores *DEFAULT* para todos as inserções no banco de dados, mesmo que esses valores gerados não façam sentido, e deixa de gerar qualquer mensagem de alerta correspondente. Esse e outros problemas são discutidos com mais detalhes no texto “MySQL ? Choose something else”[10].

Já o PostgreSQL tem uma excelente reputação com relação ao tratamento dos dados, de ser muito estável e dea gerarção de alertas, informando ao usuário todos os erros. O PostgreSQLEle faz valer as restrições que foram colocadas na criação do banco de dados e nas tabelas. Mas pelo fato de o PostgreSQL fazer todas as validações e ter todos os controles por padrão, o banco é considerado um pouco mais lento do que o MySQL, mas que para a maioria das aplicações é insignificante.

Então, optamos pelo uso do PostgreSQL no projeto.

Central de controle – Javascript framework

Um projeto de aplicação web um pouco mais complexo requer a escolha de alguma biblioteca de JavaScript, já que esta possui muitos recursos e facilidades para o desenvolvimento de web sites. Caso contrário teríamos que implementar e efetuar testes em todos os navegadores disponíveis. Sendo o JavaScript a linguagem para o desenvolvimento web, optamos por escolher uma. As facilidades que essas bibliotecas trazem são, por exemplo, manipulação do DOM[11] (Document Object Model), tratamento de eventos, chamadas Ajax[12] e animações. Além disso, permitem mostrar informações em formato de árvore, mostrar um grid (uma tabela com mais recursos) de informações, criar painéis com abas, selecionador de datas e outros recursos, que demandariam muito esforço para serem criados.

Existe uma grande quantidade de bibliotecas e frameworks JavaScript. As mais usadas são: jQuery, Prototype, Yahoo UI, Dojo, Mootools e outras. Após analisar várias dessas ferramentas, a biblioteca escolhida foi a jQuery, por ser muito difundida e com muitos tutoriais e livros a disposição para o aprendizado. O jQuery tem o menor tamanho de todas as outras bibliotecas e o foco é

manipular o DOM, enquanto que as outras tem um foco maior em expandir as funcionalidades do JavaScript. A opção por jQuery também foi influenciada pela apresentação[13] que explica as bibliotecas JavaScript mais populates e discute os seus recursos e limitações.

Central de controle – Framework de design e estrutura

O desenvolvimento web possui um importante requisito que é o de garantir um efeito visual (design e layout) idêntico para todos os navegadores e suas versões. Porque cada empresa faz a sua implementação de como a informação é apresentada na tela, gerando um trabalho maior em escrever códigos compatíveis com os diferentes navegadores. Com o avanço das tecnologias web como o HTML5 e o CSS 3.0 surgiram os CSS Frameworks[14]. Esses frameworks tem o intuito de manter o mesmo design entre os diferentes navegadores, fazendo com que se tenha poucos códigos específicos para cada navegador. Hoje existem várias opções de frameworks, como o Twitter Bootstrap, Foundation, Blueprint, Pure, entre outros.

O CSS Framework escolhido foi o Twitter Bootstrap[15] porque tem mais componentes do que os demais framewoks, é muito difundido e conta com uma excelente documentação, na forma de tutoriais, blogs e livros. O Twitter Bootstrap utiliza um sistema de grids. Em design gráfico, grid é uma estrutura com várias linhas em série que se intersectam na vertical, horizontal ou em algum outro angulo, para dar estrutura ao conteúdo. O framework Twitter Bootstrap divide o grid em 12 colunas (como pode ser visto na figura 3) que se adaptam de acordo com o tamanho da janela do navegador. Então o site é projetado no maior tamanho de grid e a medida que a janela vai diminuindo o seu conteúdo vai se adaptando ao tamanho da tela até chegar nos dispositivos móveis, que possuem telas bem pequenas. Mas para que o conteúdo se adapte é necessário que o designer faça as alterações necessárias, o framework não é capaz de decidir ~~fazer~~ sozinho como organizar o conteúdo.

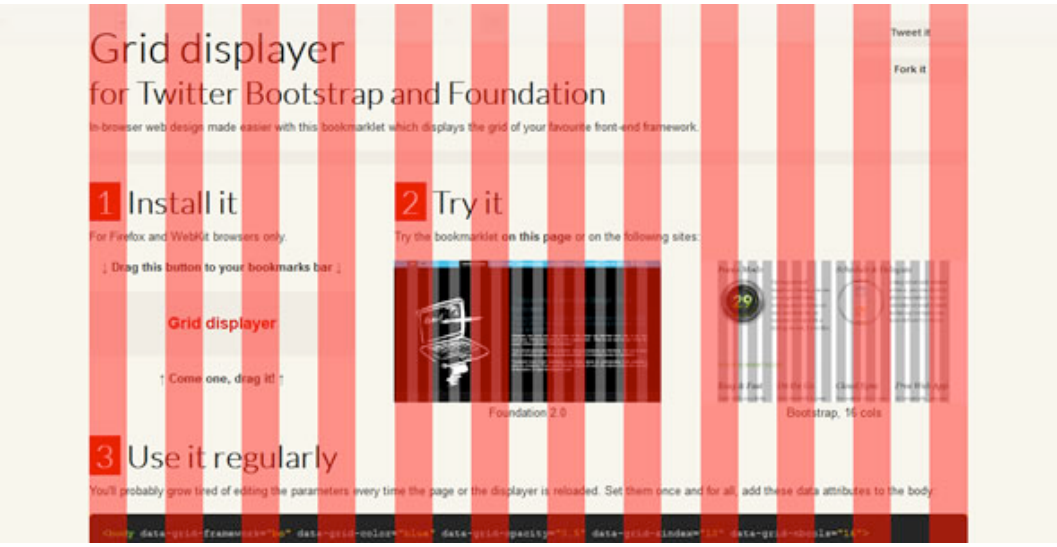

Figura 3: Visualização do sistema de grid do Bootstrap. (fonte: http://blogupstairs.com/displays-grid-framework-browsers/)

4.2 Metodologia e etapas do projeto

A metodologia do projeto do sistema foi separado nas seguintes fases~~, para facilitar o desenvolvimento e ir atacando os problemas a medidas que eles fossem surgindo, então ficou assim~~:

1) Desenvolvimento do software cliente Android para a captura dos sinais obtidos do IOIO-OTG, e envio dos sinais utilizando a ClientLib (e o MR-UDP) para a central de controle, executando em um servidor na nuvem.
2) Construção inicial da página web para a leitura dos sinais enviados, mostrando em uma janela o estado da bomba (ligado/desligado).
3) Teste de longa duração do aplicativo Android enviando os sinais para um servidor na nuvem.
4) Desenvolvimento da central de controle web para a visualização das informações recebidas, incluindo as opções de visualização em tempo real das unidades no mapa, gerenciamento das áreas geográficas e um gerenciamento básico de usuários que acessam o sistema.
5) Construção de um simulador que será integrado ao aplicativo Android para simular várias unidades (equipamentos Motofog) em movimento no mapa e enviando as informações dos sinais para a central.
6) Teste final de integração de todos os componentes e um teste em campo com 2 motos com o sistema instalado.

5. Tecnologias utilizadas

5.1 Middleware DDS

5.1.1 O que é DDS ?

O DDS[17] surgiu da necessidade de ter uma comunicação de alto desempenho e que fosse interoperável, escalável, com alta performance e em tempo real. Ele segue o modelo de *Pub/Sub centrado em dados (Data Centric Publish-Subscribe)*, ou seja, onde todos os nós são entidades que ou escrevem dados (i.e., publicadores ou *Publishers)*, ou consomem dados (i.e., assinantes o u *Subscribers)* de tópicos pré-definidos. Todas as estruturas de dados que compõem um tópico são descritas através de uma Interface Definition Language (IDL), e publicadores e assinantes interagem de forma assíncrona referenciando apenas os tópicos correspondentes, e sem a necessidade de se referenciarem mutuamente.

Sempre existiram muitos protocolos proprietários e experimentais para sistemas Pub/Sub, mas muitos deles não atendiam os requisitos de alta vazão, escalabilidade e comunicação em tempo real. Então duas empresas a norte-americana Real-Time Innovations e a francesa Thales Group, se juntaram para criar a especificação DDS, que foi submetido a OMG[18] para tornar-se um padrão. Em 2003 foi publicado a primeira versão, o DDS 1.0. Atualmente, ele se encontra na versão DDS 1.2 [19], que foi publicado em 2007.

5.1.2 Funcionamento do DDS

Sendo um middleware, o DDS opera na camada entre a aplicação e o sistema operacional fazendo a comunicação entre publicadores (produtores de dados) e assinantes (consumidores de dados) de um tópico. O DDS possui uma arquitetura decentralizada (P2P) e pode-se criar políticas de controle de qualidade (Quality of Service - QoS), fazendo com que mensagens possam ser priorizadas no envio, sejam garantidamente entregues a todos os assinantes ativos, possam ser persistidas, e assinantes possam ser notificados quando um publicador se desconectar da rede, entre outras.

O funcionamento básico do DDS tem os seguintes componentes: o **publisher** que envia dados de um ou mais tópicos, e o **subscriber** que se subscreve a um ou mais tópicos e fica à espera de dados enviados para esses tópicos. Tanto o **publisher** como o **subscriber** precisam fazer parte de um mesmo domínio DDS. Os tipos de dados de um tópico são definidos através de uma IDL, e cada tópico assim define um tipo de mensagem que trafega em um domínio DDS. Em cada domínio, cada tópico tem um nome único, um tipo de dados e um conjunto de políticas de Quality of Service (QoS), que definem como os dados serão transmitidos pelo tópico.

O **publisher** possui um componente que é o *DataWriter*, esse é responsável por escrever os tópicos no domínio, e o **subscriber** possui o *DataReader*, que é responsável pela leitura dos tópicos no domínio.

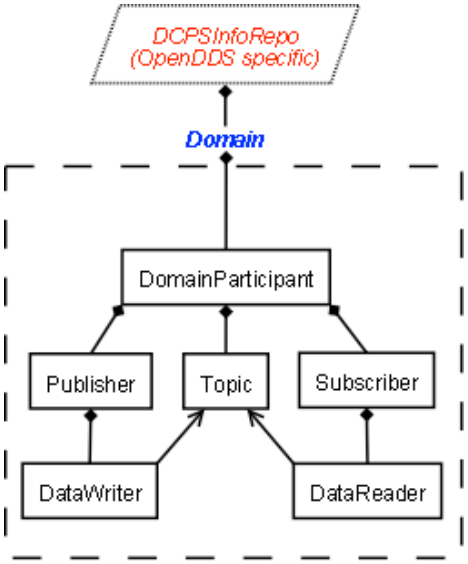

Figura 4: Estrutura básica do DDS (fonte: http://www.opendds.org/dds_overview.html)

O fluxo de dados funciona da seguinte forma: através de seu DataWriter o **publisher** publica um dado em um tópico do domínio, sendo que esse tópico precisa já ser conhecido no domínio. Esse dado é propagado na forma de uma mensagem para todos os DataReaders de **subscribers** que estão inscritos nesse tópico específico. A entrega das mensagens ocorre de acordo com as políticas de QoS previamente definidas. Por exemplo, se a politica *Reliable Delivery* estiver setada, o dado permanece armazenado no buffer do (DataWriter) Publisher até que todos os subscribers tenham recebido a mensagem. Outra QoS é a Priority Lanes, que permite definir que certos dados trafegados por um tópico devem ter maior prioridade do que outros. Ao total, o padrão DDS suporta 22 politicas de QoS[24].

5.2 Middleware SDDL

5.2.1 O que é o SDDL ?

O Scalable Data Distribution Layer é um middleware responsável pela troca de mensagens de aplicação entre nós fixos - em um cluster ou nuvem, chamado de núcleo do SDDL (*SDDL Core*) - e quaisquer nós móveis (como smart phones, tablets, notebooks) que possuam com conectividade sem fio com a Internet, por exemplo, WiFi ou 2G/3G.

O SDDL é baseado em dois protocolos: o Data Distribution Service for Real-Time Systems (DDS) [17] para a comunicação entre os nós fixos do SDDL Core, e o Mobile Reliable-UDP (MR-UDP)[20] para troca de dados entre o SDDL Core e os nós móveis. Além de prover comunicação ponto a ponto entre quaisquer dois nós (fixos ou móveis), o SDDL também possui serviços de broadcast e de groupcast, isto é, envio simultâneo para todos os nós ou para um grupo de nós. Grupo esse, que pode ser definido dinamicamente, a partir da igualdade, ou proximidade, de algum campo da mensagem de aplicação. Assim, consegue-se por exemplo, entregar uma mesma mensagem a todos os nós móveis que estejam co-localizados em uma região geográfica.

Uma aplicação distribuída baseada no SDDL essencialmente consiste de software cliente, que executa no nó móvel, e software servidor que executa em um ou mais nós do SDDL Core. Devido a alta vazão de dados e baixa latência da comunicação no SDDL Core, qualquer serviço de processamento de dados da aplicação pode ser naturalmente replicado em vários servidores no SDDL Core, tornando assim as aplicações naturalmente escaláveis.

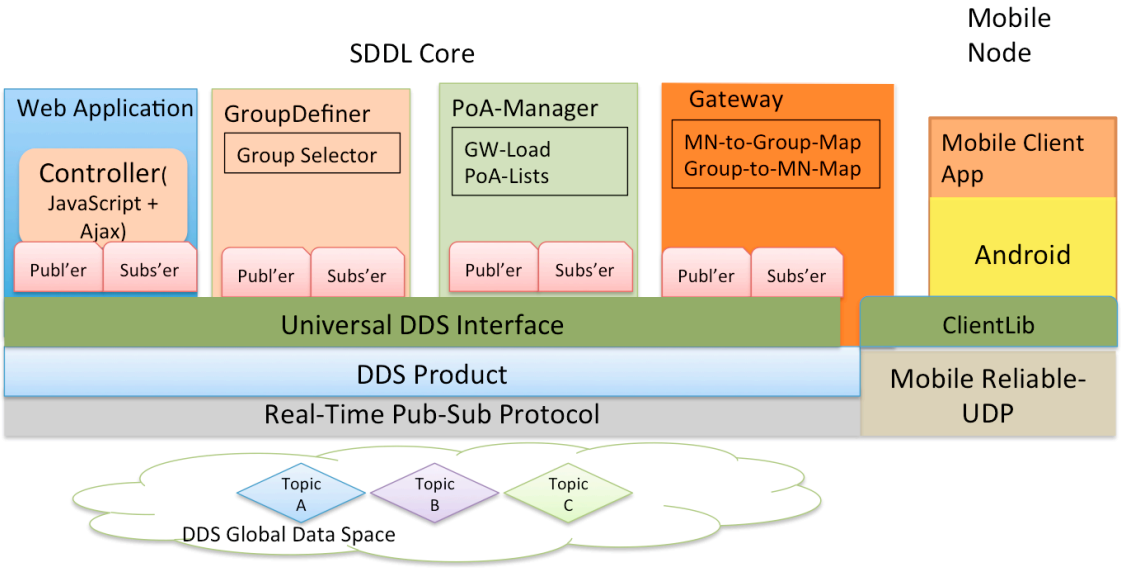

Figura 5: Visão geral da arquitetura do SDDL

5.2.2 Componentes do SDDL

A Figura 5 mostra os principais componentes e protocolos que compõe o SDDL. Dentre os componentes do SDDL Core, vários componentes são opcionais e não foram usados nesse projeto. Os únicos utilizados foram a Universal DDS Interface, que provê uma API genérica de DDS, que funciona

para vários produtos DDS, o gateway, o MR-UDP e a ClientLib, descritos a seguir.

5.2.2.1 O Gateway

O gateway é um componente que define um ponto de acesso ao SDDL Core para os nós móveis. Ele é responsável por gerenciar as conexões MR-UDP de vários nós móveis, simultaneamente, e fazer a transposição de mensagens MR-UDP para dados transmitidos através dos tópicos DDS específicos da aplicação (representados pelos losângulos na Figura 5), e vice-versa. Cada unidade móvel possui um identificador único, UUID, que é independente de seu endereço IP, porta ou MAC-Address. Além disso, todos os gateways monitoram se a conexão com cada nó móvel continua ativa (através do recebimento periódico de *heartbeats*), e se subscrevem a mensagens dos tópicos DDS da aplicação que tenham como destinatário qualquer UUID de um de "seus" nós móveis. Isso possibilita que se um nó móvel perder conexão com um gGateway, este pode se conectar a qualquer outro gGateway, e as mensagens destinadas a ele serão transparentemente encaminhadas para o novo gateway.

5.2.2.2 O MR-UDP

O *Mobile-Reliable-UDP*, ou MR-UDP é o protocolo de comunicação entre o nó móvel e o gateway. Ele implementa um UDP confiável, com algumas funcionalidades do TCP, como por exemplo confirmação de entrega de pacotes. O MR-UDP foi customizado para lidar com os problemas de conectividade intermitente, travessia de *Firewall* / NAT e troca espontânea de endereço IP em uma das pontas, por exemplo, quando um nó móvel troca de conectividade WiFi para 2G/3G.

Algumas características importantes do MR-UDP são: otimização do número de pacotes para abertura e manutenção de uma conexão ativa, monitoramento de atividade da conexão através de *heartbeats* periódicos, utilização dos UUIDs como identificadores de conexão, tentativa de re-estabelecer uma reconexão (até certo número de vezes) quando houver uma desconexão do enlace sem fio, ou uma troca de endereço IP. Transparência na continuidade de conexão, quando o nó móvel tenta enviar uma mensagem e não possui conexão, o MR-UDP armazena essas mensagens para um envio futuro assim que a conexão for reestabelecida.

Os exemplos dados anteriormente são muito importantes, porque as redes sem fio de telefonia móvel (2G/3G/4G) não são muito confiáveis, e estão sujeitas a frequentes desconexões e trocas de endereço IP. O MR-UDP é capaz de reenviar as mensagens para o destino corretamente, porque cada conexão e nó móvel possui um identificador único. Então, quando uma conexão é reestabelecida a mensagem vai para o destinatário correto.

O MR-UDP é ainda responsável pela segmentação de mensagens grandes (e a composição das mesmas a partir dos pacotes) e utiliza compressão e serialização dos dados no envio das mensagens. Isso torna o tamanho das mensagens pequenas, fazendo uma excelente utilização da rede.

5.2.2.3 A ClientLib

A ClientLib foi desenvolvida para rodar no aplicativo do cliente, no nosso caso no aplicativo Android, e utiliza o MR-UDP como protocolo de comunicação. A ClientLib possui vários recursos para facilitar o desenvolvimento da aplicação, como por exemplo o gerenciamento da lista de endereços de gateways alternativos (i.e., a PoA-List) e do handover transparente entre gateways, por exemplo, quando ocorre uma desconexão no MR-UDP, ~~t~~Toda troca de mensagens é assíncrona e é implementada através de *listeners* com métodos de call-back. Com isso, os métodos só são chamados quando ocorrer um evento de interesse para a aplicação, como por exemplo, ocorrência de uma desconexão definitiva do MR-UDP (após seguidas tentativas de reconexão) ou a chegada de uma nova mensagem, evitando a necessidade de *polling*, i.e. ficar verificando periodicamente a ocorrência do evento.

A utilização da ClientLib é bem simples: precisa-se somente implementar uma interface dos *listeners* e criar a conexão inicial com um gateway (passando com o parâmetros o seu IP e porta). A partir daí, a ClientLib cuida de todo o resto, incluindo da bufferizacão de mensagens sem confirmação de entrega, a tentativa de reconexão, o ***handover*** (de conexão) entre gateways, etc. Isso tudo diminui bastante o trabalho do programador da aplicação.

5.3 A plataforma Android

O Android[21] é um sistema operacional baseado no kernel do Linux e foi desenvolvido primeiramente para dispositivos com tela sensíveis ao toque e para telefones inteligentes e **tablets**. O sistema operacional teve seu inicio em 2005 criado pela empresa Android Inc., e acabou sendo comprada pelo Google em 2007. O Android é baseado na linguagem de programação Java, quer dizer, todo o desenvolvimento é escrito em Java, mas seu código final é compilado para outra linguagem que roda nos telefones e **tablets**. Todo o seu código é aberto, que tem como licença de uso o *Apache License*[22], mas existem alguns componentes que tem o código fechado e proprietário.

Os aplicativos Android são compostos de 4 diferentes tipos de componentes, sendo que cada um serve para um determinado propósito e tem ciclos de vida diferentes em seu momento de criação e destruição. Os quatro tipos de componentes são *Activities*, *Serviços (Services)*, *Content Providers* e *Broadcast Receivers*, que serão explicados a seguir.

Todo aplicativo quando publicado é dado um ID único. Isso faz com que cada aplicativo em momento de execução seja isolado~~s~~ dos outros aplicativos. Mas existe a possibilidade de dois aplicativos compartilharem o mesmo ID. Isso possibilita que aplicativos consigam compartilhar arquivos.

As permissões para execução de operações e acesso a recursos do dispositivo pelo aplicativo são declaradas em um arquivo de configuração em XML, o arquivo de manifesto. Dessa forma é possível mostrar ao usuário todas as ações que o aplicativo pode efetuar no dispositivo.

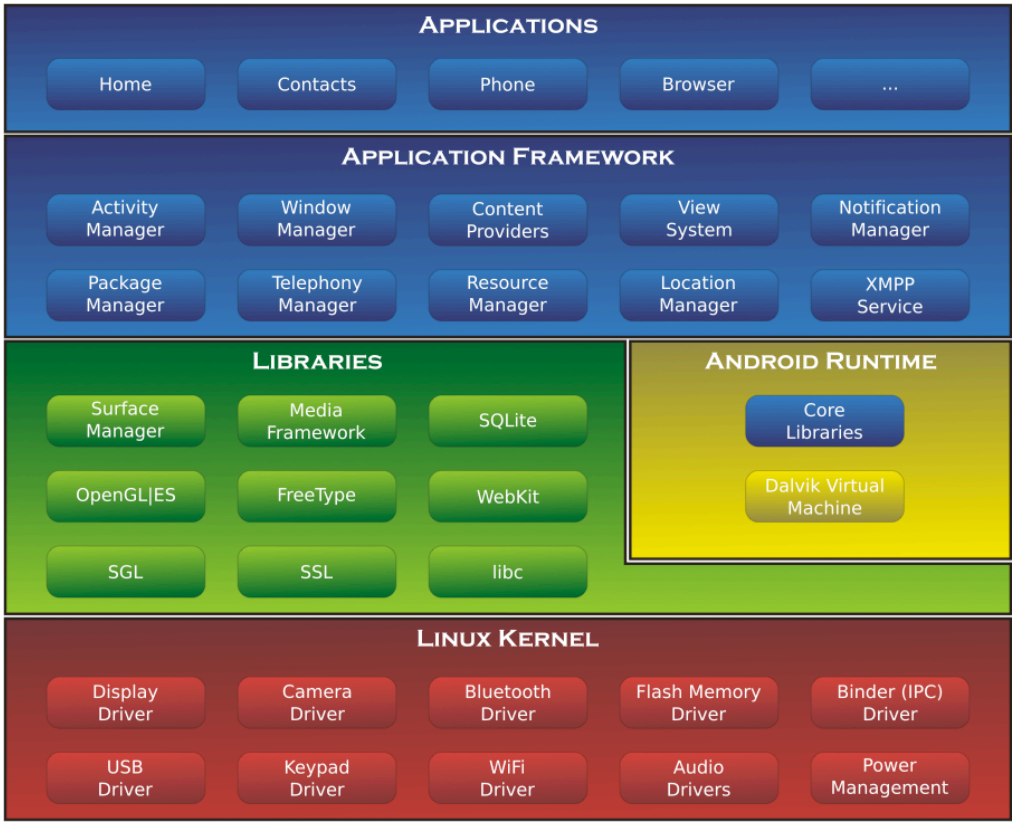

Figura 6: Arquitetura do sistema Android (fonte: http://elinux.org/Android_Architecture)

5.3.1 Activities

Uma *activity* está associada a um layout e representa uma tela da interface do usuário. Especificamente, implementa os métodos de call-back que irão ser chamados a cada ação do usuário na tela. Por exemplo, em um aplicativo de e-mail, a lista dos e-mails seria uma *activity*, a outra tela de composição do e-mail seria outra *activity*, etc. Mesmo que várias *activities* aparentem formar um grupo coeso de telas que permitem ao usuário realizar uma tarefa (por exemplo, ler e responder a um Email), na verdade elas são componentes independentes, podendo qualquer componente da própria - ou até de outra - aplicação, abrir uma *activity* (caso a permissão correspondente tiver sido dada). Por exemplo uma *activity* da câmera fotográfica, depois de tirada uma foto, pode abrir outra *activity* de outra aplicação para exibição, armazenamento, edição ou envio da foto por algum mecanismo (Email, chat, ou rede social).

5.3.2 Serviços (Services)

O serviço é um componente que roda em *background* para efetuar tarefas de longa duração. Um serviço não prove de uma interface gráfica para o usuário. Por exemplo, um serviço pode tocar música em *background* enquanto o usuário está em um *aplicativo* diferente ou o serviço pode estar obtendo dados pela rede sem bloquear a interface do usuário. A forma de iniciar um serviço pode ser através de uma *activity* ou a chamada de um *broadcast receiver*, mas existem outras formas também. Uma *activity* e um serviço podem trocar mensagens entre si, como no exemplo anterior, se um serviço estiver obtendo dados pela

rede e ele terminar o trabalho a *activity* pode ser atualizada para mostrar ao usuário de que a tarefa terminou.

5.3.3 Content Providers

O *content provider* é um componente que gerencia o compartilhamento dos dados do aplicativo. É possível armazenar dados no arquivo de sistema, em um banco de dados como o SQLite, na internet ou em qualquer outro tipo de armazenamento de arquivo que o aplicativo possa acessar. Através do *content provider* outros aplicativos podem obter ou modificar os dados do aplicativo que está sendo acessado (caso o aplicativo de direitos para isso). Por exemplo, o aplicativo dos contatos do telefone, ~~ele~~ possui um *content provider* que ~~deixa~~ permite outros aplicativos terem acesso e até modificar os contatos do telefone. Mas geralmente os *content providers* são utilizados para a leitura e a escrita de dados não compartilhados por outros aplicativos, quer dizer, ficam restritos ao próprio aplicativo.

5.3.4 Broadcast Receivers

O *broadcast receiver* possui um funcionamento muito específico, que é responder por anúncios que são enviados de outros componentes a todo o sistema Android. Normalmente esses anúncios tem origem no sistema Android. Por exemplo, anúncios de que a tela do dispositivo desligou, a bateria está com um nível baixo, uma imagem foi capturada pela câmera e assim por diante. Aplicativos podem também enviar anúncios de broadcast para o sistema, como informar que uma determinada informação foi executada. Como os *broadcast receivers* não possuem uma interface gráfica, suas ações de resposta são comumente mostradas ao usuário utilizando a barra de status do dispositivo, que fica normalmente localizada no topo. O *broadcast receiver* tem sua utilização como se fossem um “gateway”, passando a frente as atividades a serem realizadas para as activities ou serviços.

5.3.5 Local Broadcast Receivers

O *local broadcast receiver* funciona exatamente como um *broadcast receiver*, mas com uma pequena diferença. Todo anúncio feito por um aplicativo é restrito ao seu próprio aplicativo, não sendo enviado ao sistema todo. Assim o custo de processamento é menor comparado ao *broadcast receiver* e ele é mais seguro.

Uma das formas de trocar mensagens entre componentes é utilizando o *local broadcast receiver,* mas existem várias outras formas de comunicação entre componentes e processos que o Android disponibiliza.

5.3.6 Shared Preferences

O *shared preferences* é uma forma de armazenamento de dados do Android. Os dados são salvos na memória do telefone e ficam persistidos entre sessões e até mesmo quando o telefone mata o aplicativo. Os dados que podem

ser salvos precisam ser primitivos, quer dizer, os dados precisam ser *intergers*, *booleans*, *strings*, *floats e etc*. O *shared preferences* utiliza uma combinação de chave-valor, então na necessidade de se salvar uma informação o desenvolvedor precisa informar uma chave e depois o valor primitivo a ser associado a chave informada.

~~Uma das formas de persistência de dados no Android é com a utilização de *Shared Preferences*. Ele utiliza a memória interna do telefone para armazenar uma combinação de chave-valor de tipos primitivos. Essa forma de armazenamento é persistente entre sessões e até mesmo quando o aplicativo é morto pelo sistema.~~

Outras formas de armazenamento são a de memória interna, que difere da *shared preferences,* porque não se limita a chave-valores somente; Armazenamento externo, como em um cartão de memória inserido no telefone; banco de dados SQLite; e por conexão utilizando a rede.

6. O Projeto

Nessa seção, descreveremos os principais componentes do sistema de monitoramento e gerenciamento remoto do equipamento Motofog que são: O aplicativo Android que irá rodar no telefone, o serviço DDS que rodará no servidor junto com o SDDL e a central de controle (que é o website).

6.1 O Aplicativo Android

O aplicativo Android se comunica com o IOIO-OTG para a obtenção dos sinais do sistema Motofog, interage o LocationManager do Android para obter as coordenadas geográficas da posição corrente da moto, e envia esses dados para a central de controle, que processa e armazena esses dados em um servidor na nuvem, através dos componentes descritos na seção 6.2. A ~~imagem abaixo~~figura 7 mostra a arquitetura do aplicativo.

Essencialmente, o aplicativo Android consiste dos seguintes quatro serviços (*Service*): O serviço principal, *Main Controller*; o *Connection Service,* que utiliza a ClientLib para se comunicar e gerenciar a conexão com o g**ateway** do SDDL; o *Location Service* que é responsável por obtenção das posições do nó móvel via LocationManager; e o IOIO Service, que obtém os sinais do sistema Motofog, através da placa IOIO-OTG.

Foi também implementado um banco de dados local (SQLite) para armazenar os dados (sinais) a serem enviados para a Central de controle. Ele se faz necessário para garantir que dados coletados não sejam perdidos quando os mesmos não podem ser enviados para a central, por exemplo quando não é possível estabelecer nenhuma conexão MR-UDP com o gateway.

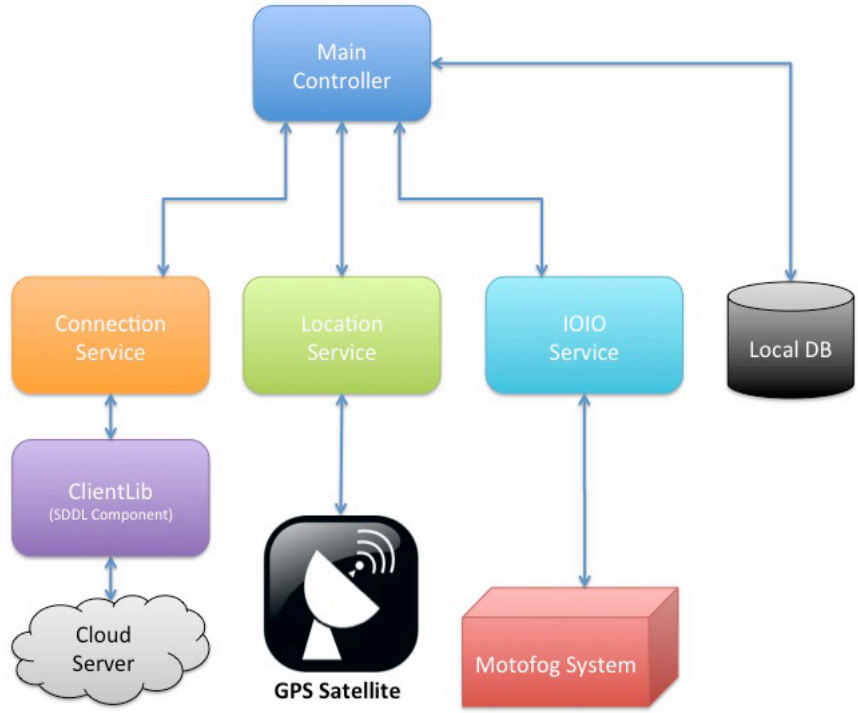

Figura 7: Arquitetura do aplicativo Android

Main Controller

O *main controller, ou controlador principal,* é o serviço principal de toda a aplicação, ele faz todo o controle dos outros serviços. Se um serviço for desligado é ele o responsável por configurar um alarme que irá disparar o reinicio do serviço. Um exemplo é o serviço do IOIO. Quando a moto for desligada ou o conector USB perder contato com a placa, o IOIO perde conexão. Então o serviço do IOIO antes de terminar envia uma mensagem para o controlador principal da desconexão e logo em seguida um alarme é configurado para o reinicio do serviço do IOIO. O alarme possui uma repetição e só é cancelado quando uma conexão é refeita com o IOIO.

Outro gerenciamento importante é o do banco de dados local. Todas as mensagens antes de serem enviadas para a central de controle, são armazenadas no banco de dados local. Caso a conexão do smartphone com o gateway tenha sido quebrada e a central ficar impossibilitada de receber os sinais coletados ainda não enviados, garantindo assim que nenhum sinal seja perdido.

O Android é um sistema que faz o gerenciamento automático de recursos. Ele tem o poder de finalizar um serviço ou um aplicativo sem aviso, então o armazenamento no banco de dados local é mais uma garantia.

O *main controller* também possui um *local broadcast receiver* para tratar as mensagens endereçadas e ele. Estas mensagens são broadcasts locais enviados pelos serviços de localização e do IOIO, contendo uma nova localização e o estado da bomba, respectivamente. O funcionamento geral do ***main controller*** segue o seguinte fluxo: os 3 serviços são iniciados, o de conexão, o Location Service e o IOIO-Service. O IOIO-Service envia o estado inicial do sistema Motofog, e o controlador principal armazena essa informação em uma variável no serviço. Em seguida, o Location Service obtém a posição corrente do LocationManager, e envia ao controlador. Com essas duas informações é criado um evento chamado *UnitEvent*, que contém todas as

informações necessárias, como o UUID (o identificador do dispositivo), a latitude e longitude, o estado da bomba e etc, que é enviado a Central de Controle utilizando o Connection Service. Cada *UnitEvent* só é enviado quando se tem uma conexão MR-UDP estabelecida, mas todo *UnitEvent* é também armazenado no banco de dados local, e uma flag indicará, para cada evento armazenado, se este já foi enviado com sucesso para a Central de Controle. Será necessário uma limpeza do banco SQLite, porque os dispositivos possuem uma memória interna muito pequena. Então de 6 em 6 horas ou de 12 em 12 horas será disparado um alarme que irá efetuar a limpeza dos eventos enviados. Mas o tempo do alarme é totalmente configurável.

Connection Service

O Connection Service, ou serviço de conexão, como o próprio nome diz, utiliza a ClientLib para estabelecer e manter uma conexão com o gateway. Ele possui uma lista de todos os *UnitEvents* a serem enviados ao servidor, e pode funcionar de duas formas diferentes: envio imediato de cada evento (UnitEvent) recebido, ou envio periódico de lotes de UnitEvents, por exemplo, a cada 2 minutos, sendo que esse período é configurável. Ao enviar as mensagens o serviço de conexão informa ao controlador principal da última mensagem enviada. Essa última mensagem contém o último evento enviado, assim o controlador principal altera a flag no banco local de todos os eventos anteriores a esse, criando o controle de quais eventos foram enviados ou não.

Outra funcionalidade importante é com relação a perda de conexão e o reestabelecimento de uma conexão perdida. Toda vez que uma conexão é estabelecida com o gateway, o serviço de conexão informa ao controlador principal que está conectado. Este faz então uma busca no banco de dados local para verificar se existem eventos a serem enviados. Em caso positivo, o controlador principal envia ao Connection Service os *UnitEvents* que precisam ser transmitidos.

Location Service

O Location Service, ou serviço de localização é responsável pela obtenção periódica da localização corrente do nó utilizando o Android LocationManager no modo GPS. O intervalo para a obtenção de um nova localização é configurável. No nosso caso, o Location Service está configurado para a obtenção de novas localizações a cada 30 segundos. Essa frequência da obtenção das posições foi escolhida, porque atende bem os requisitos de precisão da informação geo-localizada da aplicação. Além disso, como o smartphone normalmente estará sendo alimentado pela energia do equipamento Motofog (através do IOIO-OTG), em princípio não existe o problema da falta de energia pela utilização continuada do GPS, que demanda uma cota significativa de energia para funcionamento, mesmo após sincronização com os 4 satélites, em média, aprox. 143 mW, segundo[29].

Porém, quando o sistema Motofog estiver desligado, o smartphone será alimentado apenas pela energia de sua bateria. Então, enquanto existir energia na bateria o Location Service vai estar ativo, causando uma rápida drenagem da energia da bateria, causando o eventual desligamento do dispositivo. Então, quando o motor da moto for novamente ligado, o sistema Motofog e o IOIO-OTG voltarão a receber energia, o smartphone deverá se ligar automaticamente, reiniciando todos os serviços. Para que isso seja possível, foi necessário fazer o *jailbreak* do telefone, como alterar alguns arquivos do sistema para fazer com

que o telefone ligue ao receber energia. O *jailbreak* é o processo de remover as proteções do telefone e ter acesso a todos os arquivos do sistema. Agora, para fazer com que o telefone ligue automaticamente quando uma fonte de energia é recebida, foi alterado a parte em que o telefone reconhece que está sendo carregado. Então em vez de manter o telefone desligado, um comando do tipo "/system/reboot" é inserido no arquivo do sistema. Fazendo com que o telefone ligue ao receber energia e não fique desligado carregando somente a bateria.

IOIO Service

O IOIO-Service é responsável pela leitura dos sinais do sistema Motofog, e para isso utiliza a API para acesso e controle da placa IOIO-OTG.

Um ponto importante a ser mencionado é com relação a eventual perda de conexão do IOIO-Service com a placa IOIO-OTG, situação muito estudada e testada. Quando ocorre uma desconexão com a placa IOIO-OTG, devido a uma falha de energia (moto morrer), desconexão do cabo USB por trepidação ou etc, é enviado um local broadcast ao controlador principal, que então configura um alarme (usando o Android Alarm Manager) com um tempo pré-determinado para reiniciar o IOIO-Service e reestabelecer a conexão com o IOIO.

Esse procedimento funcionou muito bem em todos os testes de desconexões realizados em laboratório, por exemplo, desligamento e religamento do sistema Motofog simulado. Em todos os casos, o IOIO-Service sempre conseguia fazer a reconexão com a placa IOIO-OTG com sucesso.

Local DB

O Local DB é uma instância de banco de dados SQLite, disponibilizado no sistema Android. O banco possui uma única tabela com as seguintes colunas: a identificação (auto-incremento) de cada tupla (INTEGER), para selecionar individualmente cada uma delas; a que marca o tempo (TIMESTAMP~~timestamp~~) de criação do *UnitEvent*; as informações do evento (i.e. o estado da bomba Motofog e a coordenada GPS) serializados utilizando a biblioteca GSON[25] em JSON[23] (TEXT), e por último a coluna com a *flag (INTEGER),* para indicar se o evento foi ou não enviado à Central de Controle (onde "0" informa que o evento não foi enviado).

6.2 Os Serviços no Servidor

O Servidor é uma máquina de uma infra-estrutura de nuúvem, que executa três processos: um servidor Apache, o *SDDL Server* e o gateway. Enquanto os primeiros dois processos interagem compartilhando um banco de dados, no caso o PostgreSQL, os últimos dois são componentes do SDDL, que por sua vez utiliza o DDS OpenSplice Community Edition[28]. A Arquitetura geral é mostrada na Figura 8.

Figura 8: Arquitetura dos serviços no servidor

A primeira parte, a do DDS, que na figura 8 é mostrado na parte da direita, são os serviços do SDDL Server e o do gateway. O gateway é o componente que o telefone conecta e é por onde é recebido os sinais no servidor. Ao receber o sinal o gateway via DDS repassa ao SDDL Server, que insere no banco de dados. A segunda parte, que é o serviço do Apache, é responsável por toda a parte da web. Os usuários tem acesso a central de controle utilizando o navegador que conecta ao Apache. Toda e qualquer informação que o usuário queira, o Apache consulta o PostgreSQL~~banco de dados~~ que é repassado ao navegador.

6.2.1 O Serviço do DDS e o SDDL Server

O SDDL Server é o componente que recebe as mensagens (com as coordenadas geográficas e os sinais do Motofog) vindas do gateway, que por sua vez recebe os dados do Aplicativo Android através do MR-UDP. O SDDL Server verifica se as mensagens são do tipo *UnitEvent*. Caso sejam, insere todas as informações no PostgreSQL~~banco de dados~~, para a persistência dos eventos.

O SDDL Server foi construído para que os parâmetros de conexão com o banco de dados e o vendedor DDS usado seja lido de um arquivo de configuração. É possível ter no arquivo de configuração vários parâmetros, assim só é preciso informar no inicio do arquivo qual parâmetro irá ser utilizado.

Um conjunto de configurações é composto por: O IP do servidor, qual o tipo do banco de dados, se é PostgreSQL, MySQL ou outro qualquer, qual o vendedor DDS (CoreDX, RTI ou OpenSplice), e os parâmetros como nome do banco e etc, e são definidos no arquivo. A esse conjunto de parâmetros é atribuído um nome para a identificação.

Exemplo de um arquivo de configuração:

```
[default]
params=postgresql93_localhost
dds_vendor=OpenSplice

[postgresql93_localhost]
jdbc_driver=postgresql
username=postgres
password=
ip_address=192.168.1.10
port=5432
schema=fumajet

[postgresql93_remote]
jdbc_driver=postgresql
username=postgres
password=
ip_address=172.80.15.8
port=5432
schema=fumajet
```

Como pode ser observado no exemplo do arquivo, existem duas configurações possíveis, a **postgresql93_localhost** e a **postgresql93_remote**, a que está sendo usada no momento fica na declaração do **params** na secão **[default]**.

Abaixo do **params** vem declarado qual o é produto DDS que será utilizado. No exemplo acima é o **OpenSplice**, da PrismTech. O SDDL provê suporte para 3 produtos DDS, o CoreDX[26], o RTI Connext[27] e o OpenSplice Community Edition[28].

6.2.2 O Serviço Web

O serviço web roda o Apache com a linguagem PHP, para processamento dos dados a serem visualizados no navegador. Esse serviço é responsável por manter a central de controle em operação. Ele obtém as informações do banco de dados que é alimentado pelo SDDL Server.

Os outros componentes que fazem parte do serviço da web são: O framework Laravel que é feito em PHP; O Twitter Boostrap que é o responsável pelo design da página; E o jQuery que faz toda a comunicação via Ajax com o servidor.

O acesso do navegador ao servidor Web é feito por requisições HTTP e por chamadas Ajax. Assim, usando requisições Ajax, partes da central de controle podem ser atualizadas sem a necessidade de carregar a página completa. Essa forma de operação otimiza bastante a utilização da rede (conexão com o servidor na nuvem), bem como economiza recursos de memória e processamento do servidor.

6.3 O Website da Central de Controle

A central de controle é o componente que permite o monitoramento das motos com o equipamento Motofog. O monitoramento inclui tanto a visualização da posição das motos no mapa em tempo real, como a visualização das cercas eletrônicas (e áreas geográficas pré-cadastradas no sistema). As ~~áreas~~cercas eletrônicas, onde o produto do fumacê deve ser aplicado, podem ser ativadas ou desativadas no momento de visualização. De fato, não existe qualquer vinculação implícita entre cada motocicleta com uma ~~área~~ cerca eletrônica cadastrada para aplicação do produto: qualquer motocicleta poderá ser despachada para qualquer uma das ~~áreas~~cercas eletrônicas. Essa não vinculação garante à empresa Fumajet uma maior flexibilidade na operação de sua frota de motos com o sistema Fumajet. Além disso, o website da Central de Controle também inclui a função de geração de relatório, com dados extraídos do banco de dados, tais como distâncias percorridas por cada moto, vazão específica da bomba Motofog a cada momento, se a aplicação do produto ocorreu em uma área (cerca eletrônica) específica da aplicação e etc.

Na versão atual do website existe também um controle básico de cadastro de usuários. Qualquer usuário pode cadastrar um outro usuário no sistema, mas a remoção de um usuário somente pode ser feita pelo usuário “root”. Esse simples controle foi implementado para prevenir abusos e problemas futuros de funcionários.

Portanto, o website possui três visões, controladas por abas, que são: monitoramento (em tempo real), o gerenciamento das áreas e o gerenciamento das unidades Motofog. No futuro serão incluídos também unidades para uso por pedestres - o Fumacê Costal. O Costal é um equipamento que é montado nas costas do funcionário que aplica os produtos, como se fosse uma mochila.

A seguir, serão descritas em maiores detalhes cada uma das visões do website da Central de controle.

Monitoramento

O monitoramento mostra em tempo real as motos que estão em campo. O mapa reflete todas as motos que ligaram o sistema no dia corrente. Ou seja, essa visão mostra as motos ativas do dia, e não as utilizadas em dias anteriores.

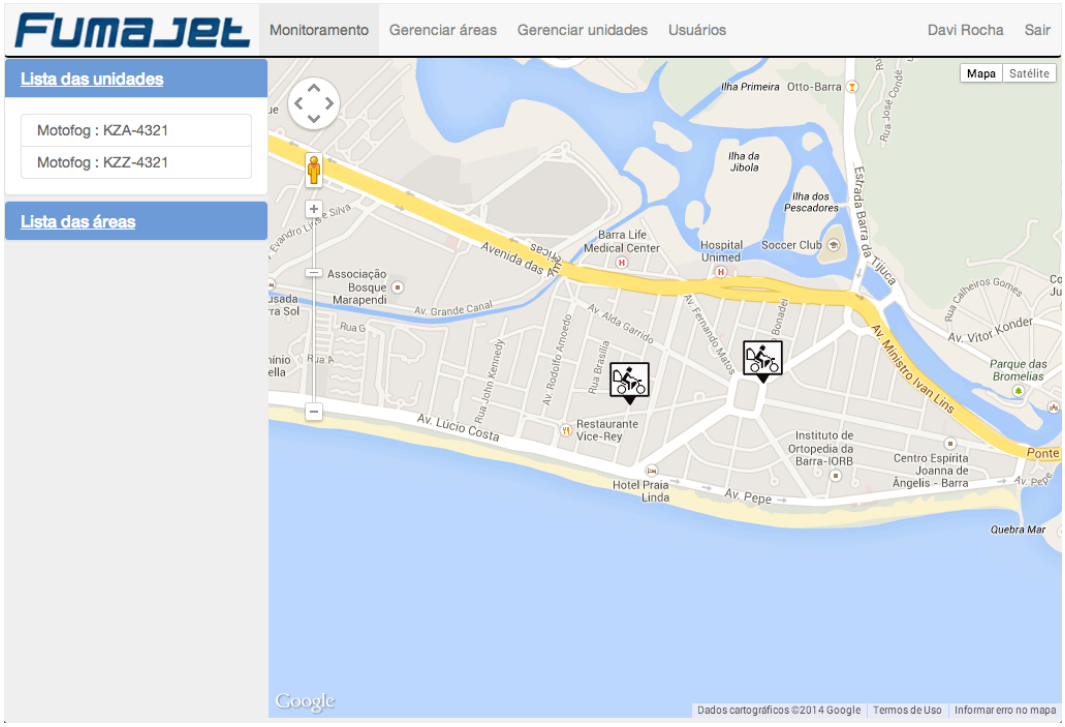

Figura 9: Monitoramento das motos quando estão em campo com a lista das unidades visível

Como pode ser observado na figura 9, na parte da esquerda da página inclui duas opções de visualização: a lista das unidades Motofog cadastradas no sistema e a lista de áreas cadastradas no sistema.

A lista das unidades apenas reflete os equipamentos Motofog cadastros no sistema, não incluindo nenhuma funcionalidade adicional.

A lista das áreas de aplicação é mais importante. Nela, são mostradas as ~~áreas~~ cercas eletrônicas cadastradas no sistema. Quando nenhuma ~~área~~ cerca eletrônica é selecionada na barra lateral (ao clicar no nome de uma cerca eletrônica~~área~~ ela é selecionada), os ícones das motos ficam todos em branco (mostrado na figura 9), informando somente a sua localização no mapa. Entretanto, quando existe uma ~~área~~ cerca eletrônica selecionada, todo o estado do funcionamento do monitoramento é alterado. Ao selecionar uma ~~área~~ cerca eletrônica da lista (a esquerda), ela será mostrada no mapa e em decorrência será avaliado se existe uma moto com a bomba ligada ou desligada dentro da área clicada. Caso a bomba Motofog esteja desligada dentro de uma área o ícone da moto fica amarelo, mas caso a bomba esteja ligada o ícone será verde (mostrado na figura 10). Dessa forma, o operador da Central de Controle terá uma visão clara se a moto está com a bomba ligada ou desligada dentro da ~~área~~ cerca eletrônica e poderá decidir se o funcionário está operando o sistema Motofog em sua motocicleta corretamente.

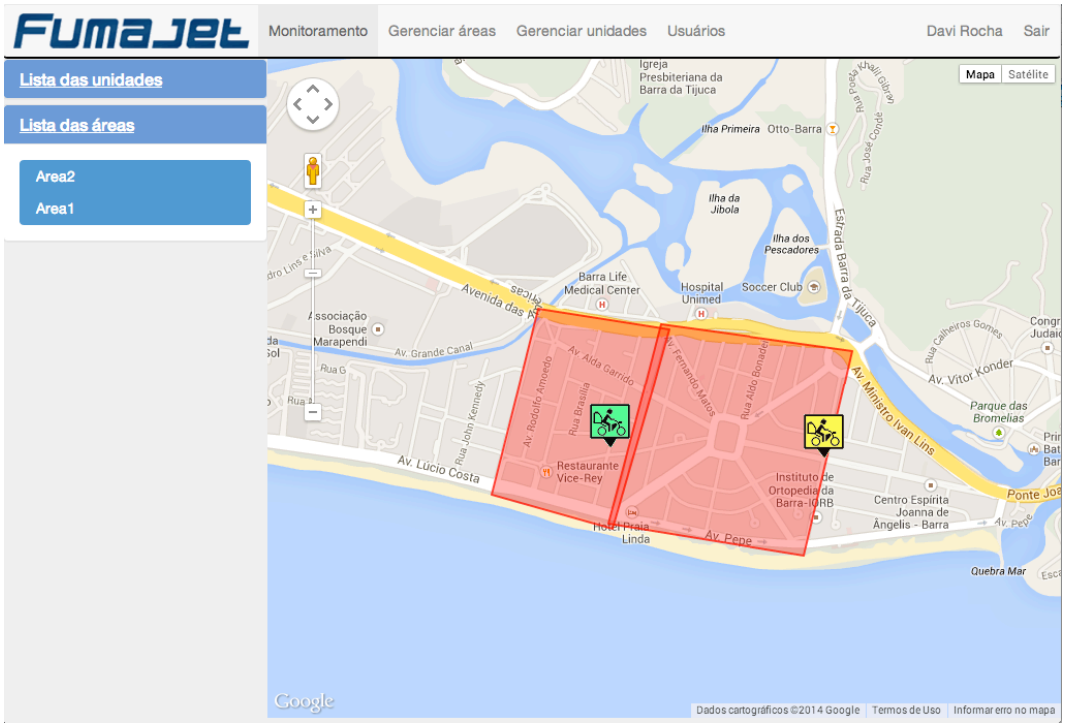

Figura 10: Monitoramento com duas ~~área~~ cercas eletrônicas selecionadas sendo mostrado o estado da bomba dos Motofogs

Como já dito, nenhuma área de aplicação estará associada a uma moto. Portanto, o operador terá que saber quais ~~áreas~~ cercas eletrônicas ele precisará habilitar/selecionar no dia. Isso dará maior flexibilidade no gerenciamento da frota de Motofogs. Por exemplo, se uma moto específica foi escolhida para aplicar o produto em uma ~~área~~ cerca eletrônica em um dia marcado, mas ocorrer algum problema e não for~~i~~ possível a moto ir no local, então não será necessário que o operador informe o problema. Ele simplesmente poderá despachar outro equipamento Motofog para ~~a área~~aquela mesma cerca eletrônica, ou então programar a aplicação (com a ativação da ~~área~~ cerca eletrônica correspondente) em outro dia.

Gerenciamento das áreas

Nessa visão, o operador é capaz de cadastrar e visualizar ~~áreas~~ as cercas eletrônicas de aplicação do produto, em quantidade arbitrária. Ao finalizar o cadastro de uma ~~área~~ cerca eletrônica ele logo verá a nova ~~área~~ cerca eletrônica na lista de áreas, na esquerda. Posteriormente poderá clicar em cada uma das ~~áreas~~ cercas eletrônicas (da lista) para a visualização de seu perímetro no lado direito da tela. Ao clicar em uma ~~área~~ cerca eletrônica já criada o site mostra na tela a área e faz um zoom mostrando a ~~área~~ cerca eletrônica selecionada, como mostrado na figura 11. Nesse caso, a ~~área~~ cerca eletrônica clicada é a com o nome de "Area1".

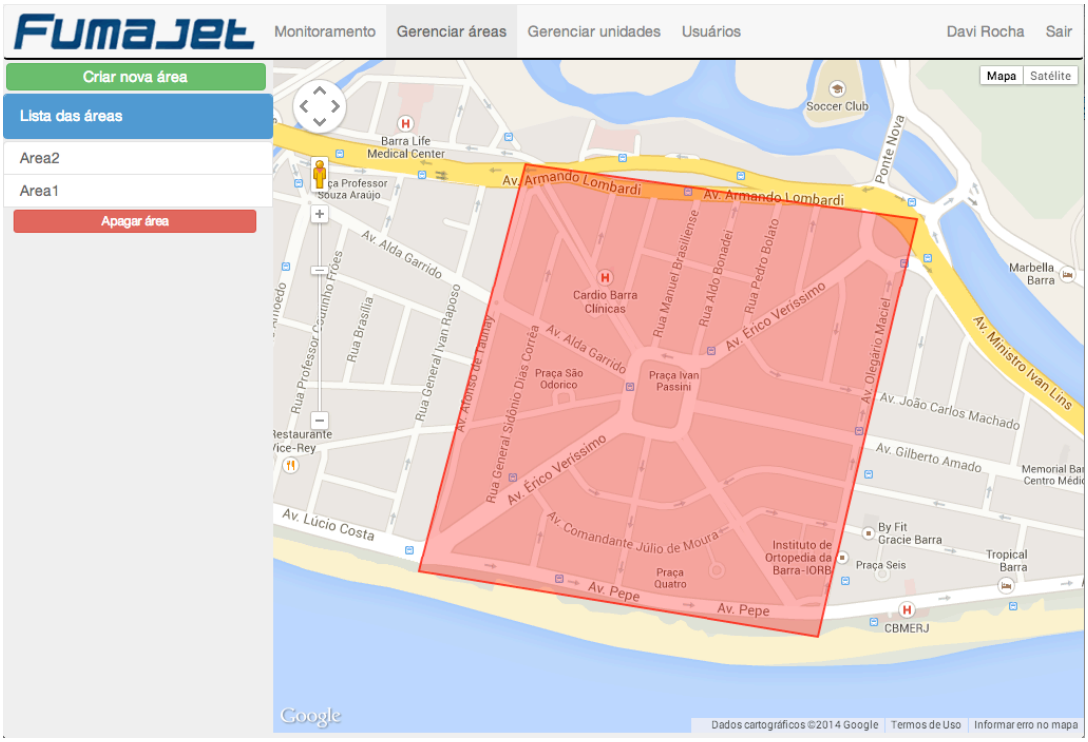

Figura 11: Setor de criação e visualização das ~~áreas~~cercas eletrônicas

O nome das ~~áreas~~ cercas eletrônicas podem ser qualquer nome ou frase que aceite o encoding UTF-8.

Ao criar uma ~~á r e a~~ cerca eletrônica é necessário fornecer duas informações: o nome ~~da área~~ (inserido em um textbox) e a outra é o perímetro a área, cujo formato deve ser um polígono, e cujos vértices são marcados diretamente no mapa (lado direito). Caso alguma dessas duas informações não sejam fornecidas, uma ~~área~~ cerca eletrônica não é criada. ~~Áreas~~ As cercas eletrônicas não podem ser modificadas. Caso precise-se modificar uma ~~área~~cerca eletrônica, é preciso remover a anterior e criar uma nova ~~área~~cerca eletrônica. Para remover uma ~~área~~ cerca eletrônica é necessário clicar no nome da ~~área~~ cerca eletrônica e em seguida clicar no botão **Apagar** em vermelho. O botão **Apagar** só é visível quando uma ~~área~~ cerca eletrônica é clicada.

Gerenciamento das unidades

Na visão de gerenciamento das unidades é mostrada uma tabela com todas as unidades cadastradas no sistema e seus atributos. Nessa versão atual aparecem somente unidades Motofog, mas no futuro a tabela incluirá também unidades do equipamento do tipo Costal.

Um ponto importante é como é feito o cadastro das unidades no sistema e como essas unidades são descobertas pelo sistema.

Para identificar de forma unívoca e precisa os dispositivos dos usuários, é comum a prática de usar número do celular do usuário, o endereço da interface de rede (NIC), ou qualquer outra informação que seja específica de cada aparelho como identificação do mesmo. Mas de acordo com Reto Meier (apresentação no Google I/O de 2011, sobre Android Protips), essa prática de se obter um identificador é não recomendada. A razão é que usuário pode trocar de aparelho, e então as informações não seriam mais relativas ao usuário em

questão. Além disso, apesar do sistema Android prever um método do tipo getDeviceID(), cada fabricante de smartphone implementa essa função de forma diferente. Por exemplo, getDeviceID() pode retornar NULL, o que compromete esse método de obtenção do identificador. Uma alternativa melhor é de gerar um identificador único, como por exemplo um UUID (Universally Unique Identifier), para cada instalação do aplicativo em um aparelho.

~~Um~~ UUID é um padrão da ISO/IEC (nr. 11578:1996) criado para identificar ~~globalmente~~univocamente um dispositivo ou máquina~~s~~ em um sistema~~s~~ distribuído~~s~~. ~~O sentido aqui de globalmente é de~~Univocamente nesse contexto significa que todos os identificadores gerados sejam únicos, ~~mundialmente~~globalmente. Dada a grane quantidade de bits de um UUID, ~~A~~a probabilidade de colisão de dois UUIDs gerados é quase nula, sendo portanto um excelente identificador para os dispositivos instalados nos Motofogs. ~~octeto composto por 16 dígitos alfanuméricos, e.g., **550e8400-e29b-41d4-a716-446655440000**. Com um ID tão grande, a probabilidade de uma colisão de nomes é muito baixa. Além disso, o~~O SDDL também faz uso do UUID como identificador de qualquer nó móvel.

Quando o aplicativo Android é instalado em uma unidade e os serviços são inicializados, é feita uma verificação para ver se já existe um UUID armazenado no dispositivo. Caso não haja, um novo UUID é gerado e armazenado no dispositivo utilizando Shared Preferences. ~~Dessa forma temos como identificar as motos de forma correta e confiável.~~

Já com um UUID próprio, quando o aplicativo Android é ligado pela primeira vez, este começa a enviar as coordenadas geográficas e o estado do sistema Motofog para a central de controle com esse UUID. A central de controle percebe então que os eventos recebidos são de um Motofog com um UUID novo, para o qual ainda não existe nenhum nome associado ao UUID, como por exemplo, a placa da motocicleta. Assim, aparece uma barra no topo da janela do website da central de controle alertando que uma nova unidade foi detectada e que precisa ser associado a um nome, a fim de completar o cadastro efetivo da unidade Motofog ao seu UUID. Na figura 12 é mostrada a barra de alerta mostrando que uma nova unidade que foi identificada pelo sistema.

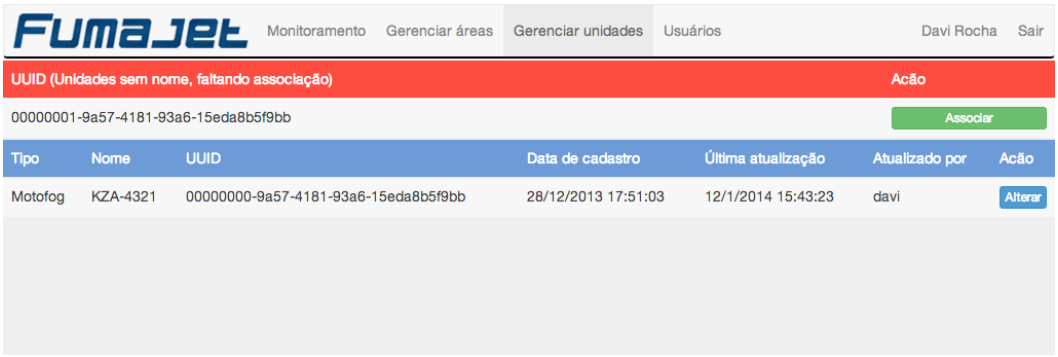

Figura 12: Nova unidade detectada pelo sistema que ainda não possui um nome associado

Na figura 12, pode ser observado também que logo abaixo da barra de alerta, aparece a lista das unidades que já foram cadastradas no sistema, com seu tipo, Nome, UUID, data de cadastro, etc.

Para associar um nome a uma nova unidade basta clicar no botão verde **Associar** (como mostrado na figura 12), e no formulário preencher os campos necessários e concluir o cadastro. No exemplo da figura 13, o nome dado a essa unidade é **KZZ-4321** e o tipo é um **Motofog**.

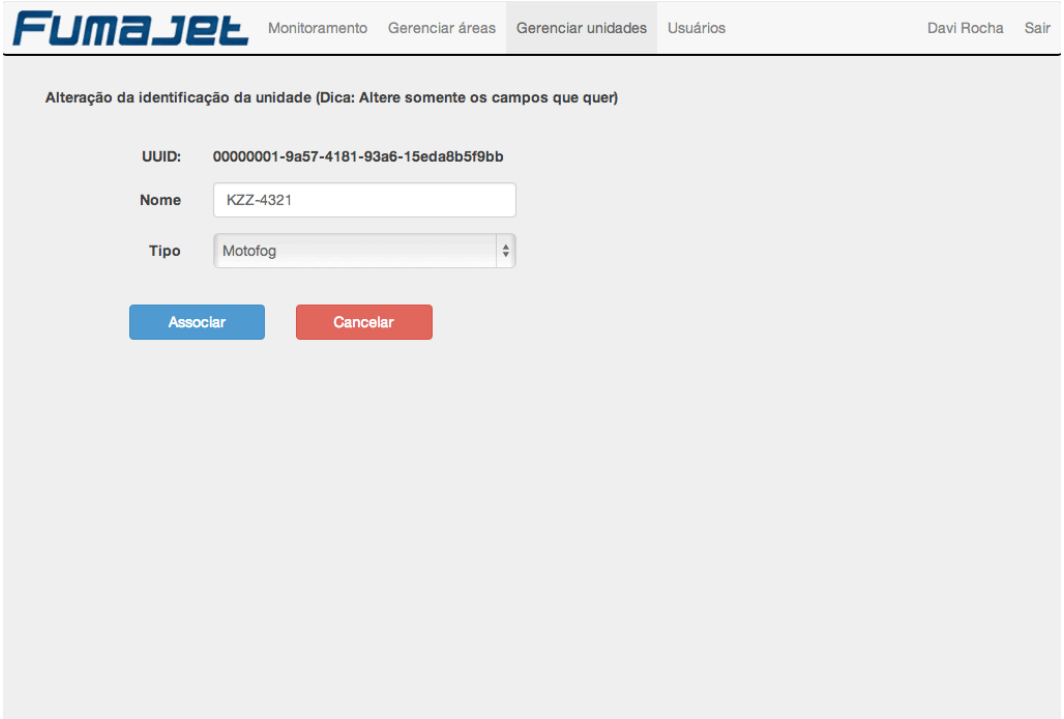

Figura 13: Exemplo de uma nova unidade sendo cadastrada no sistema

Ao completar o seu cadastro (associando o nome), a nova unidade aparece na lista de unidades cadastradas e agora pode ser visualizada na lista da visão monitoramento (vide seção 6.3.1).

7. Testes realizados

7.1 Teste do middleware SDDL e do aplicativo Android

Um dos componentes do SDDL utilizado no projeto é o Ggateway ~~. O gatewayque~~ utiliza o protocolo MR-UDP para a comunicação sem fio com os smartphones. ~~Então já que o gateway é o ponto de comunicação com os telefones móveis~~Dada a sua importância no SDDL, foram feitos vários testes de conexão e desconexão com o Ggateway, descritos~~. Os testes do gateway serão explicador~~ a seguir.

Um simulador foi criado para auxiliar nos testes. O simulador simula o GPS e ~~d~~o estado da bomba. No simulador existe a opção de alterar o estado da bomba de ligado para desligado, e vice-versa. Os dois eventos são enviados para o controlador principal, criando o UnitEvent para ser enviado ao servidor utilizando o serviço de conexão.

A bateria de testes do SDDL e do aplicativo Android começou com o teste do gateway.

O primeiro teste foi executado com o gateway sendo terminado enquanto s e possuía uma conexão estabelecida, quer dizer, o gateway está operando normalmente e o telefone móvel enviando os sinais para o servidor. Em seguida o processo do gateway no servidor é finalizado, esperando para ver se o serviço de conexão do telefone móvel reconhece ~~de~~ que houve uma desconexão com o ~~servidor~~Gateway. No teste pôode ser observado que alguns listeners não foram chamados corretamente, então o telefone achava que a conexão com o gateway

ainda estava estabelecida. O problema foi resolvido com os listeners e agora o telefone consegue detectar uma desconexão com o gateway.

O próximo teste do gateway foi terminar o gateway com uma conexão estabelecida e logo em seguida iniciar o gateway novamente. O mesmo cenário do exemplo anterior foi utilizado, mas a única diferença foi que o Ggateway depois de alguns segundos foi iniciado novamente. Esse teste foi feito para ver se o protocolo MR-UDP ~~conseguisse~~conseguiria recuperar a ~~reconectar~~conexão após uma desconexão. O protocolo conseguiu refazer a conexão com o Ggateway.

No aplicativo Android os serviços foram testados. No caso do serviço de localização o teste não chegou a ser executado, porque ~~toda parte de~~o acesso ao GPS ~~pertence~~faz parte da plataforma Android e durante os testes dos outros serviços foi verificado que o GPS sempre funcionou corretamente.

O teste final do aplicativo foi verificar se os eventos estavam sendo salvos corretamente no banco de dados local (SQLite). Com o simulador ligado, os eventos são gerados e o controlador principal armazena os eventos no SQLite. Nesse momento é feito uma verificação na tabela que armazena os eventos, e pôde ser constatado que os eventos estavam sendo armazenados corretamente. Logo em seguida os eventos são passados ao serviço de conexão e ~~consequentemente~~assim enviados ao servidor. Nesse segundo momento o SQLite é consultado novamente para verificar se os eventos enviados tiveram as suas *flags* alteradas para refletir o envio. A alteração das *flags* no SQLite foi executada corretamente.

Mas um problema ~~chamou a atenção~~foi detectado~~. No~~: no caso de perda de conexão com o Ggateway, os sinais estavam sendo enviados em duplicidade. Isso estava ocorrendo, porque ao reestabelecer uma conexão com o Ggateway, o serviço de conexão envia uma mensagem ao controlador principal perguntando se existe algum evento a ser enviado. O serviço de conexão possuí uma lista de eventos a enviar. Então o controlador principal enviava os eventos com a flag indicando que não tinham sido ~~de não~~ enviados, e esses eventos já estavam na lista de envio no serviço de conexão, gerando assim uma duplicidade nos eventos armazenados no servidor. Esse problema ~~sendo detectado~~ foi corrigido e agora somente um evento de cada sinal gerado chega ao servidor.

7.2 Teste da placa IOIO e do IOIO Service

Os testes ~~da~~de acesso à placa IOIO, ~~no caso o hardware, foi~~foram feitos pelo Marlon Moura, ~~o engenheiro responsável~~o desenvolvedor do hardware. Um dos testes críticos foi~~é~~ o do serviço IOIO de perda de conexão, explicado a seguir.

No ~~serviço do~~ IOIO Service precisava ser testada~~o~~ a questão da perda de conexão ~~com~~entre a placa IOIO e o sistema Android. ~~Foi possível fazer~~Assim, foram feitos - vários testes de desconexão e reconexão com a placa. Por exemplo, com o sistema todo ligado e operando normalmente, o equipamento Motofog foi tirado da tomada, simulando assim uma moto que “morreu” no meio da rua. Ao religar o equipamento o serviço conseguiu reestabelecer a conexão com a placa e voltou a operar normalmente. Os testes de desconexão e reconexão foram feitos exaustivamente e na maioria dos casos houve uma reconexão com a placa. Pequenos ajustes foram feitos durante os testes para a correção de pequenos problemas. Mas no final dos testes a desconexão e reconexão está bastante estável.

7.3 Teste da central de controle (Website)

A central de controle teve todas as suas seções do site testadas. Os formulários foram os primeiros a ser testados: ~~. No caso~~ os formulários eram preenchidos e enviados ao servidor e depois era feito a verificação no PostgreSQL para ~~saber~~verificar se tinham sido inseridos corretamente no banco de dados.

As duas seções que ~~tiveram~~demandaram mais correções foram a seção de monitoramento e a do ~~o~~ gerenciamento das áreas (cerca eletrônica).

Em sua primeira versão, ~~A~~a seção de monitoramento ~~no inicio dos testes~~ estava duplicando os ícones no mapa, g~~. G~~erando assim uma poluição no mapa de ícones, mas que logo em seguida foi corrigida.

Na seção de gerenciamento das áreas foi observado que quando uma área ~~quando~~ era desenhada no mapa, os ~~seus~~ pontos do polígono correspondente eram enviados ao servidor ~~iriam~~ fora de ordem. Quando uma cerca eletrônica era selecionada para visualizar o seu desenho final, podia ser observado que estava completamente ~~errado~~dirfente d~~o~~a área originalmente desenhada~~original~~. Para solucionar esse problema o modo de envio dos pontos ao servidor foi modificado:~~,~~ ~~o operador~~ a medida que o operador ia marcando os pontos, esses ~~pontos~~ eram sendo armazenados em ordem de criação em um vetor. Então ao enviar o formulário para o servidor os pontos eram enviados na ordem de criação, de maneira que, ~~e~~ a o ~~clicar~~selcionar uma área para ser ~~visualizar~~visualizada ~~a cerca eletrônica~~, ~~a área desenhada estava~~esta era exibida de acordo com a original.

8~~7~~. Considerações finais

Na ausência de um sistema completo que atendesse às necessidades de operação em campo de equipamentos móveis Motofog da empresa Fumajet, foi desenvolvido um sistema customizado para monitoramento e gerenciamento utilizando um smartphone Android acoplado ao sistema do Motofog.

O sistema de hardware e software apresentados nesse trabalho foram concebidos de forma a poder, futuramente, incorporar novas funcionalidades e agregar valor ao produto Motofog da Fumajet. Esse projeto tratou apenas dos componentes de software: o aplicativo Android, o middleware SDDL, e o Servidor Apache com PHP.

8~~7~~.1 O que aprendi com esse trabalho

Esse projeto para a empresa Fumajet foi para mim um projeto extremamente interessante e valioso, pois forçou-me a aprender e ganhar experiência prática em muitas novas tecnologias e sistemas de várias áreas da computação.

Em particular, aprendi a fundo programar para o sistema Android, e para diversas tecnologias web envolvidas, como JavaScript, HTML, CSS. Tive também que aprender a configurar o middleware DDS e a programar usando toda a tecnologia desenvolvida no LAC, como a UDI e a ClientLib do SDDL, com o seu enfoque de comunicação assíncrona Pub/Sub. Além disso tive que

configurar um banco de dados como o PostgreSQL, e aprendi também um pouco sobre hardware (acesso ao IOIO-OTG), bem como configurar servidores na nuvem.

Então, considero que trabalhei em um projeto completo que me deu uma visão geral do desenvolvimento de um sistema bem extenso e complexo, gerando em mim grande satisfação e prazer em concretizar tudo.

8.2 Próximos passos

O sistema ainda tem muito o que melhorar. Tem a parte de criação de relatórios que será feita em seguida. Basicamente, será mais uma visão do website, na qual o operador poderá selecionar uma área de aplicação e uma unidade Motofog, e o sistema irá mostrar a distância que a unidade percorreu naquela área. Além disso, pretendo também implementar a visualização do trajeto da moto no mapa, ao longo do tempo, facilitando a criação de relatórios específicos por intervalo de tempo.

Os códigos JavaScript serão mais modularizado do que atualmente, diminuindo a redundância de código.

Além disso, faz-se necessário um controle melhor de acesso dos operadores ao sistema, criando mais opções de controle. Assim o sistema deverá ficar mais seguro e evitar que o operador da central de controle seja capaz de fazer graves erros ao utilizar o sistema.

Por fim, reconheço a necessidade de um re-design do layout e estilo do website, que foi feito apenas para mostrar as funcionalidades, uma vez que esse não foi o enfoque desse projeto.

98. Referências Bibliográficas

[1] Sítio do Projeto Motofog, http://www.fumajet.com.br/motofog

[2] L. David, R. Vasconcelos, L. Alves, R. Andre, G. Baptista, and M. Endler, "A Communication Middleware for Scalable Real-Time Mobile Collaboration," *2012 IEEE 21st Int. Work. Enabling Technol. Infrastruct. Collab. Enterp.*, pp. 54–59, Jun. 2012.

[3] Sítio do Zend Framework, http://framework.zend.com/

[4] Sítio do CodeIgniter, http://ellislab.com/codeigniter

[5] Sítio da EllisLab, http://ellislab.com/

[6] Sítio do CakePHP, http://cakephp.org/

[7] Sítio do Laravel, http://laravel.com/

[8] Sítio do MySQL, http://www.mysql.com/

[9] Sítio do PostgreSQL, http://www.postgresql.org/

[10] Texto explicativo sobre o porque não escolher o MySQL como banco de dados para um aplicativo, http://grimoire.ca/mysql/choose-something-else

[11] Documento que explica o que é o DOM (Document Object Model) e o porque, http://www.w3.org/DOM/

[12] Ajax, também chamado de Asynchronous JavaScript and XML, http://en.wikipedia.org/wiki/AJAX

[13] Apresentação sobre as diferentes bibliotecas de Javascript, talk feito por John Resig na Ajax Experience de 2008, http://www.slideshare.net/jeresig/javascript-library-overview-presentation/

[14] Explicação do que é um CSS Framework, http://en.wikipedia.org/wiki/CSS_frameworks

[15] Sítio do projeto do Bootstrap, http://getbootstrap.com/

[16] Sítio do IOIO do Google, https://github.com/ytai/ioio/wiki

[17] Pardo-Castellote, G. (2003). OMG data-distribution service: architectural overview. In *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings*. (pp. 200–206). IEEE. doi:10.1109/ICDCSW.2003.1203555

[18] Sítio da OMG (Object Management Group), http://www.omg.org/

[19] DDS, versão 1.2, Janeiro de 2007, http://www.omg.org/spec/DDS/1.2

[20] L. David, M. Roriz and M. Endler. "MR-UDP: Yet another Reliable User Datagram Protocol, now for Mobile Nodes", Monografias em Ciência da Computação, 6/13, ISSN: 0103-9741, Abril 2013

[21] Sítio do Android, http://www.android.com/

[22] Android Overview, Open Handset Alliance, http://www.openhandsetalliance.com/android_overview.html

[23] JavaScript Object Notation, a lightweight data-interchage format. http://www.json.org/

[24] RTI, "RTI Data Distribution Service - Comprehensive Summary of QoS Policies," 2011 http://community.rti.com/rti-doc/45e/ndds.4.5e/doc/pdf/RTI_DDS_QoS_Reference_Guide.pdf (visitado em janeiro 2014)

[25] Projeto do Google de serialização de objetos Java em JSON (JavaScript Object Notation) chamado GSON. http://code.google.com/p/google-gson/

[26] Sítio do projeto do CoreDX. http://www.twinoakscomputing.com/coredx

[27] Sítio do projeto do RTI. https://www.rti.com/products/dds/

[28] Sítio do projeto do OpenSplice. http://www.prismtech.com/opensplice

[29] A. Carroll, G. Heiser, An Analysis of Power Consumption in a Smartphone, Proceedings of the USENIX Annual Technical Conference, 2010.