UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
ESPECIALIZAÇÃO EM TECNOLOGIA JAVA

WAGNER PARISOTO

SISTEMA INTEGRADO PARA GEOLOCALIZAÇÃO E RASTREAMENTO

MONOGRAFIA

PATO BRANCO
2015

WAGNER PARISOTO

SISTEMA INTEGRADO PARA GEOLOCALIZAÇÃO E RASTREAMENTO

Trabalho de Conclusão de Curso, apresentado ao III Curso de Especialização em Tecnologia Java, do Curso de Especialização em Tecnologia Java, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Especialista.

Orientador: Prof. Robison Cris Brito.

PATO BRANCO

2015

SISTEMA INTEGRADO PARA GEOLOCALIZAÇÃO E RASTREAMENTO

Por

Wagner Parisoto

Esta monografia foi apresentada às 15h30 do dia 28 de outubro de 2015 como requisito parcial para a obtenção do título de ESPECIALISTA, no III Curso de Especialização em Tecnologia Java, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. O acadêmico foi arguido pela Banca Examinadora composto pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Prof. Msc. Robison Cris Brito
Orientador
UTFPR – Campus Pato Branco

Prof. Msc. Rubia Eliza de Oliveira Schutz Ascari
Banca
UTFPR – Campus Pato Branco

Profa. Dr. Beatriz Borsoi
Banca
UTFPR – Campus Pato Branco

Prof. Msc. Robison Cris Brito
Coordenador do curso de Especialização
UTFPR – Campus Pato Branco

AGRADECIMENTOS

Os meus sinceros agradecimentos a todos que me auxiliaram na elaboração deste trabalho, especialmente a minha família pelo apoio dado, ao meu orientador prof. Robison Cris Britto por ter dedicado seu tempo a fim de acompanhar o desenvolvimento do trabalho.

RESUMO

PARISOTO, Wagner. **Sistema Integrado para Geolocalização e Rastreamento**. Pato Branco, 2015. 58 f Monografia. Especialização em Tecnologia Java. Universidade Tecnológica Federal do Paraná, Campus Pato Branco. Pato Branco, 2015.

Este Trabalho de Conclusão de Curso, apresenta a abordagem utilizada para o desenvolvimento de um sistema cliente-servidor que oferece suporte para geolocalização e rastreamento de pessoas por meio de dispositivos móveis. O sistema, quando instalado no dispositivo móvel de uma pessoa, permite por meio de alguns mecanismos, como por exemplo, satélites artificiais, capturar a sua localização em qualquer ponto do globo terrestre e posteriormente, por meio de uma conexão com a Internet, enviar os dados para um servidor remoto, permitindo assim o acompanhamento em tempo real da pessoa e também o seu histórico de localização.

Palavras-chave: Android. Java. GPS. Geolocalização.

ABSTRACT

PARISOTTO, Wagner. **Integrated System for Location and Tracking**. Pato Branco, 2015. 58 f Monograph. Expertise in Java technology. Federal Technological University of Paraná, Campus Pato Branco. Pato Branco, 2015.

This work Completion of course, presents the approach used for the development of a client-server system that supports geolocation and tracking of people through mobile devices. The system when installed on the mobile device of a person, allows through some mechanism, such as artificial satellites, capturing its location anywhere on the earth and then through a connection to the Internet, send data to a remote server, thus allowing real-time monitoring of the person and also your location history.

Keywords: Android. Java. GPS. Geolocation.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta uma descrição do estudo do negócio para o qual o software foi desenvolvido, bem como funcionalidades e características gerais.

1.1 Considerações Iniciais

Com o crescente aumento nas vendas dos dispositivos móveis nos últimos anos, tais como *tablets* e *smartfones*, hoje um número significativo da população mundial já possui um destes dispositivos, segundo a União Internacional de Telecomunicações (UIT, 2015), já somam cerca de 7 bilhões de dispositivos em todo o mundo.

As vendas de smartfones no Brasil durante o segundo trimestre de 2014 somaram 13 milhões de aparelhos comercializados, um crescimento de 22% em relação ao mesmo período de 2013, de acordo com estudo da empresa Internacional Data Corporation (IDC, 2014) . Ainda segundo a empresa, em abril e junho, foram vendidos mais de 100 *smartfones* por minuto. É possível observar na pesquisa realizada pela empresa, que o número de vendas de *smartfones* está muito alto, e a maioria dos telefones celulares vendidos atualmente são desse tipo.

Estes dispositivos já não possuem poder de processamento tão limitados se comparados aos primeiros aparelhos celulares, é comum encontrar recursos como envio e recebimento de dados multimídias a partir dos aparelhos, câmeras fotográficas, tela de alta resolução, sensores e em especial, o *Global Positioning System* (GPS).

Assim como os celulares evoluíram para os *smartfones* e *tablets*, a rede de dados fornecida pela operadora evoluiu muito. O primeiro serviço de redes de terceira geração foi implantado no Brasil por volta de 2004 e de lá para cá o serviço evoluiu muito, chegando hoje a população brasileira ter cerca de 94% de cobertura do sinal (ANATEL, 2015).

Além da possibilidade de utilizar a rede da operadora, hoje é comum em

vários locais públicos existirem equipamentos que oferecem o acesso à rede *Wi-fi* gratuitamente, sendo esta uma alternativa barata e rápida, mesmo que limitada a poucos metros quadrados.

Desta forma, é possível utilizar as vantagens dos *smartfones*, unidas aos recursos de redes para desenvolver aplicativos cada vez mais complexos, e com um custo cada vez menor. Entre as possibilidades, está o desenvolvimento de rastreadores, que permitem identificar o local exato onde se encontra um smartfone ou *tablet* e compartilhar esta informação na Internet.

1.2 Objetivos

A seguir são apresentados os objetivos gerais e específicos abordados neste trabalho.

1.2.1 Objetivo Geral

Desenvolver um sistema que permita realizar o rastreamento e acompanhe o histórico de localização de um smartfone ou *tablet* Android equipado com GPS e uma rede de dados.

1.2.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- Permitir que uma pessoa (usuário final), crie uma conta de usuário no sistema;
- Possibilitar que o usuário mantenha cadastros de dispositivos;
- Permitir que o usuário acompanhe o histórico de localização geográfica

de um determinado dispositivo móvel;

- Permitir a geração de gráficos estatísticos que apresentem o total percorrido num determinado período.

1.3 Justificativa

Quando as pessoas saem pelas ruas existe o receio de por algum motivo perderem seu celular, ou mesmo de serem assaltadas. Ou ainda os pais tem curiosidade de saber por onde seus filhos andam, ou quem sabe o proprietário de uma empresa querer em um exato momento saber onde está o funcionário “João” com o veículo da empresa. Como saber onde está aquele determinado dispositivo móvel? Esse é um problema muito recorrente em no cotidiano das pessoas. A aplicação desenvolvida como resultado deste trabalho apresenta uma solução, para localização de celulares ou dispositivos que possuem GPS.

O sistema tem por finalidade suprir uma demanda encontrada principalmente nas empresas mas também requisitada para uso pessoal e segurança privada. A proposta visa oferecer uma solução eficiente no que diz respeito ao rastreamento e acompanhamento em tempo real da geolocalização de dispositivos móveis e pessoas, permitindo por exemplo, se um aparelho for perdido, que seja possível recuperá-lo com maior facilidade, ou mesmo obter a localização de uma pessoa e o histórico de localização.

2 REFERENCIAL TEÓRICO

Neste capítulo é apresentado o referencial teórico sobre sistemas de localização, tecnologias de desenvolvimento como Java e Android e bancos de dados orientado a objetos, também são descritos os principais conceitos a respeito de sistemas distribuídos.

2.1 GPS (*Global Positioning System*)

GPS (*Global Positioning System*) é um sistema de posicionamento global, formado por uma constelação de satélites artificiais que circundam a órbita terrestre a fim de fornecer um sistema de localização global, amplamente utilizado nos dias atuais. Muito utilizado na navegação marítima e aeronáutica para facilitar a localização hoje vem sendo popularizado a cada dia e as pessoas estão utilizando para se localizar no trânsito ou simplesmente em dispositivos portáteis como celulares.

O GPS consiste em um sistema com 24 satélites (mais alguns sobressalentes), que completam uma órbita em torno da terra a cada 12 horas em trajetórias distintas, a uma altura de 20.200 km e transmitindo pulsos de rádio em intervalos de tempo precisos (LONGLEY *et al*., 2011). Um receptor GPS é um pequeno computador que recebe os sinais dos satélites e transforma-os em informações de geolocalização (FONTANA, 2002). Existem receptores GPS que conseguem obter apenas a latitude e longitude e outros que além dessas duas ainda podem obter a altitude. A maioria dos smartfones vendidos atualmente possuem embarcado um receptor de GPS em sua estrutura o que possibilita desenvolver aplicações que utilizem a localização disponibilizada pelo receptor.

2.2 Banco de Dados Orientado a Objetos

BDOO (Os Bancos de Dados Orientados a Objetos) surgiram na década de 80, visando aumentar a produtividade do desenvolvimento de software (NASSU; SETZER, 1999).

Um BDOO é um banco em que cada informação é armazenada na forma de objeto, que só podem ser manipuladas por métodos definidos pela classe a qual o objeto pertence. De acordo com Elmasri e Navathe (2005), existem pelo menos dois fatores que levam a adoção desse modelo, são eles:

- Nos Bancos de Dados relacionais se torna difícil trabalhar com dados complexos, já que todas as informações são armazenadas por meio de tabelas relacionadas e a junção das informações é considerada relativamente complexa;
- Aplicações são construídas em linguagens orientadas a objetos e o código precisa ser traduzido para uma linguagem que o modelo de banco de dados relacional entenda, o que torna essa tarefa muito tediosa. Essa tarefa também é conhecida como “perda por resistência”.

O modelo Orientado a Objetos (OO) ganhou espaço nas áreas como banco de dados espaciais, telecomunicações e nas áreas científicas. Isso porque essa tecnologia oferece aumento de produtividade, segurança e facilidade de manutenção. Como objetos são modulares, mudanças podem ser feitas internamente, sem afetar outras partes do programa.

Segundo Khoshafian (1999, p.335), “devido a seus atributos orientados a objetos, os bancos de dados orientados a objetos oferecem modelos diretos e intuitivos para o desenvolvimento de aplicações.”

Conforme explica Nassu e Setzer (1999, p.75), o modelo utilizado para representação da estrutura de um banco orientado a objetos é OER (*Object Entity-Relationship*). E “esse modelo de dados é uma extensão do Modelo de Entidades e Relacionamentos”.

Todos os desenvolvedores das linguagens orientadas a objetos sabem das

dificuldades de passar um modelo orientado a objetos para uma persistência relacional. Grande parte do tempo de desenvolvimento de um software é empregada na fase de mapeamento das classes. Utilizando um banco de dados orientado a objetos é possível eliminar ferramentas e códigos para o mapeamento objeto relacional e aproveitar os benefícios do paradigma orientado a objetos sem estar preso pelo banco de dados. Nesse cenário, um banco de dados relacional[1] não traz nenhuma vantagem. Talvez a escolha de um modelo Orientado a Objeto seja uma evolução natural.

2.2.1 Características

Uma das grandes vantagens de um BDOO de acordo com Nassu e Setzer (1999), é que ele permite salvar objetos grandes e depois obter a recuperação facilmente desses grandes objetos como texto longos, imagens etc. Eles são considerados não estruturados porque o BD não conhece a sua estrutura. A aplicação pode utilizar várias funções para manipular esses objetos. E o mais importante é que o BD não conhece essas funções, mas através de técnicas oferecidas por ele é capaz de reconhecer esses objetos e buscá-los no banco de dados.

É importante frisar que um BDOO não é capaz de processar diretamente condições de seleções e outras operações desses objetos. É necessário que esses dados sejam passados para o BD para que ele possa saber tratar os objetos corretamente. Por exemplo, considerando objetos que são imagens. Supondo que a aplicação precise selecionar a partir de uma coleção de tais objetos somente aqueles que incluem certo padrão. Nesse caso, o usuário deve fornecer o programa de reconhecimento do padrão, como um método em objetos do tipo imagens. O BDOO recupera, então, um objeto do banco de dados e aplica nele o método para o reconhecimento do padrão para determinar se o objeto adere ao padrão desejado.

---

1 Banco de dados Relacionais,são banco de dados onde sua estrutura é baseada em registros relacionados e organizados em tabelas. Essas relações tornam os registros integrados (FERRARI, 2007);

2.3 Sistemas Distribuídos

Sistemas Distribuídos consistem em uma coleção de computadores independentes que se apresentam ao usuário como um sistema único e coerente. Esta definição tem dois aspectos. Primeiro trata do *hardware*: as máquinas são autônomas. Segundo trata do software: os usuários pensam do sistema como sendo um único computador, porém ambos são essenciais (TANENBAUM; STEEN, 2006, p.2).

Pode-se dizer também que um Sistema Distribuído é um sistema no qual os componentes de hardware e software estão localizados em computadores interligados por uma rede, comunicam e coordenam suas ações somente por meio da troca de mensagens (COULOURIS; DOLLIMORE; KINDBERG, 2013).

O motivo que se leva a construir sistemas distribuídos vem da necessidade de compartilhar informações ou recursos de *hardware* e *software* pela rede, permitindo assim que esses recursos estejam disponíveis para qualquer lugar em qualquer dispositivo, desde que conectados por meio de uma rede física e utilizando procedimentos definidos de comunicação.

Conforme explica Coulouris, Dollimore e Kindberg. (2013) para a construção de sistemas distribuídos podem ser encontrados muitos desafios, como:

- **Heterogeneidade:** um mesmo sistema distribuído pode ser construído para funcionar nos mais variados tipos de redes de computadores, sistemas operacionais, hardware e linguagens de programação. Os protocolos de comunicação da Internet mascaram as diferenças existentes nas redes e o *middleware*[2] pode cuidar das outras diferenças.

- **Sistemas abertos:** os sistemas distribuídos devem ser extensíveis,

2 ***Middleware*** ou mediador, é uma camada de software cujo objetivo é mascarar a heterogeneidade visa fornecer um modelo de programação conveniente para os programadores de aplicativos. É composto por um conjunto de processos ou objetos, em um grupo de computadores, que interagem entre si de forma a implementar a comunicação e oferecer suporte para compartilhamento de recursos a aplicativos distribuídos (COULOURIS; DOLLIMORE; KINDBERG, 2013);

ou seja, permitir que outros componentes de software sejam acoplados ao sistema, no entanto como os componentes podem ser escritos por diversos programadores, a integração se torna um desafio real.

- **Segurança:** a criptografia pode ser usada para proporcionar proteção adequada para os recursos compartilhados e para manter informações sigilosas em segredo, para quando são transmitidas por uma rede.

- **Escalabilidade:** um sistema distribuído é considerado escalável, quando consegue permanecer ativo de forma eficiente, possuindo um número considerável de recursos e usuários.

- **Concorrência:** a presença de múltiplos usuários em um sistema distribuído, possibilita que sejam requisitados acessos aos recursos de forma simultânea.

- **Transparência:** tem por objetivo tornar certos aspectos da distribuição abstratas para o programador de sistemas, para que ele se preocupe somente com os procedimentos que sua aplicação irá implementar. Por exemplo, ele não precisa estar preocupado com a sua localização, ou como as mensagens serão trocadas pela rede, se uma falha por alguma razão possa vir a ocorrer, ela poderá ser tratada como forma de exceção.

2.4 Soquetes

Soquete ou “*Socket*” no termo Inglês é o modelo mais utilizado para o desenvolvimento de aplicações distribuídas, que utiliza uma rede de computadores para a comunicação de vários dispositivos que estão interligados por ela. A comunicação costuma ocorrer por meio de uma mensagem de solicitação do cliente

enviada para o servidor, pedindo para que alguma tarefa seja executada (COULOURIS; DOLLIMORE; KINDBERG, 2013).

Um Soquete representa um ponto de conexão em uma rede TCP/IP[3], semelhante em termos de analogia com os soquetes elétricos em uma casa que provêm uma conexão ponto para os eletrodomésticos. Quando dois computadores desejam “conversar”, isto é, trocar mensagens, cada usuário usa um Soquete. Um computador denominado o servidor, abre Soquetes e espera por conexões. O outro computador denominado o cliente, chama o servidor de Soquetes para iniciar a conexão. Para estabelecer uma conexão, tudo o que é necessário é um endereço do servidor de destino e um número de porta.

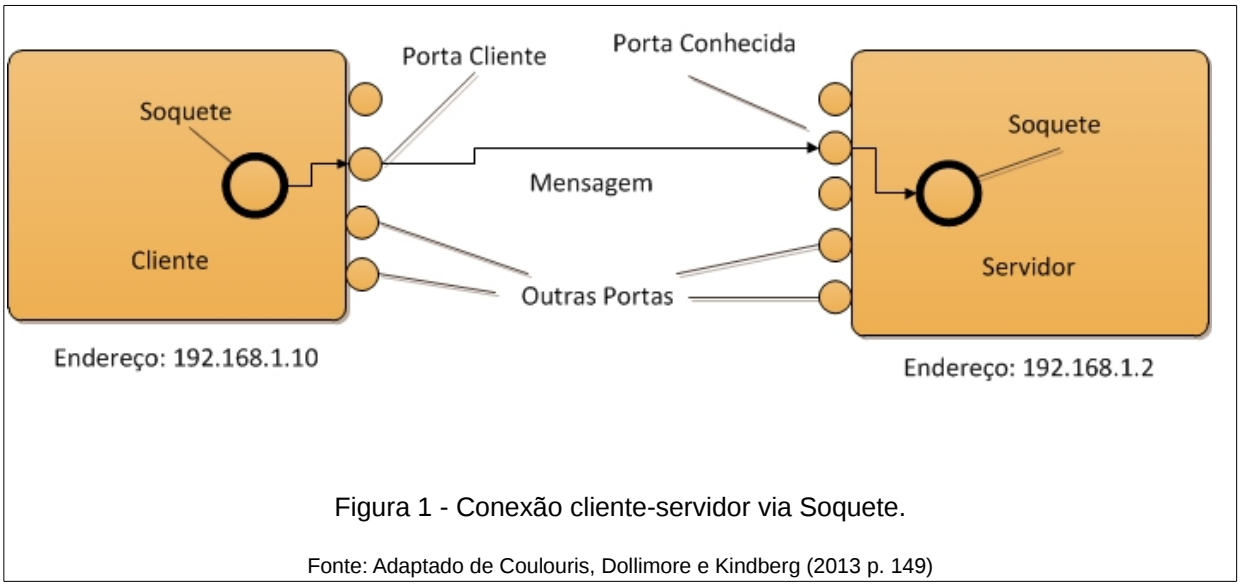

Figura 1 - Conexão cliente-servidor via Soquete.

Fonte: Adaptado de Coulouris, Dollimore e Kindberg (2013 p. 149)

Conforme visto na Figura 1, cada computador em uma rede possui um único endereço. Portas representam conexões individuais dentro daquele endereço.

Tomando como exemplo o serviço de correio de cartas, para uma carta chegar ate o seu destino (casa) o carteiro irá percorrer toda a rua até encontrar a casa que contenha determinado número (equivale a porta), como a rua possui o mesmo nome do início ao fim, a única forma de individualizar o seu destino é pelo

3 O **TCP/IP** é um conjunto de protocolos de comunicação entre computadores em rede (também chamado de pilha de protocolos TCP/IP). Seu nome vem de dois protocolos: o TCP (*Transmission Control Protoco*l - Protocolo de Controlo de Transmissão) e o IP (*Internet Protocol* - Protocolo de Interconexão). Os protocolos TCP/IP possibilitam que qualquer par de *hosts* se comuniquem, apesar das diferenças na arquitetura física (COMER; DROMS, 2007, p.268).

número da casa, dessa forma o carteiro encontrará o destino correto e entregará a carta. Cada porta dentro de um computador compartilha o mesmo endereço, mas os dados são encaminhados dentro de cada computador pelo número da porta.

Quando um soquete é criado, ele deve ser associado com uma porta específica - esse processo é conhecido como ligando a uma porta.

Soquetes possuem dois modos principais de operação (COULOURIS; DOLLIMORE; KINDBERG, 2013):

- **Modo orientado à conexão:** Soquetes orientados à conexão operam como um telefone: eles devem estabelecer uma conexão e ao final desligá-la. Tudo o que fluir entre esses dois eventos chega na mesma ordem em que foi enviado.

- **Modo não orientado à conexão:** Soquetes não orientados à conexão operam como o correio: a entrega não é garantida, e múltiplos pedaços da correspondência podem chegar em uma ordem diferente daquela em que foram enviados.

2.4.1 *WebSockets*

Os *WebSockets*, de maneira simplificada, são os tradicionais soquetes implementados de maneira um pouco diferente para que seja possível utilizá-los na Web. De acordo com a comunidade mundial que contribui para o projeto, o *WebSocket* define uma API regida pela W3C e incluindo também o seu próprio protocolo de comunicação descrito pela RFC 6455 (FETTE; MELNIKOV, 2011).

O *WebSocket* estabelece conexões de "soquete" entre um navegador da web ou qualquer aplicativo cliente e um servidor trafegando dados de forma bidirecional. Em outras palavras, há uma conexão persistente entre o cliente e o servidor e ambas as partes podem começar a enviar dados a qualquer momento. A conexão é feita por meio do protocolo TCP, utilizando geralmente a porta 80, o que facilita a utilização e implementação, pois é a porta padrão para acesso a web, e

não é bloqueada diretamente no *firewall*, porém nada impede de ser utilizada outra porta.

Os *WebSockets* são muito úteis para quando se deseja trocar informações entre um aplicativo cliente, como por exemplo um aparelho celular ou por um navegador web juntamente com um servidor de aplicação, pois a comunicação ocorre de maneira muito simples e rápida e está sendo utilizada em larga escala na web da geração atual.

3 MATERIAIS E MÉTODOS

Neste capítulo são descritos os materiais, métodos e tecnologias utilizadas para o desenvolvimento deste trabalho.

3.1 Materiais

Na implementação do sistema foram utilizados uma série de materiais, como podem ser vistos a seguir.

3.1.1 Java

Java segundo Oracle (2015), é uma linguagem computacional completa, adequada para o desenvolvimento de aplicações baseadas na rede Internet, redes fechadas ou ainda aplicações locais e para equipamentos portáteis, como telefones celulares. Atualmente está licenciada pela GNU (*General Public Licence*), sendo uma linguagem de programação de código livre e gratuita.

Foi desenvolvida por volta de 1995 nos laboratórios da empresa *Sun Microsystems* com o objetivo de ser mais simples e eficiente do que suas linguagens predecessoras. O alvo inicial era a produção de software para produtos eletrônicos de consumo (fornos de micro-ondas, agendas eletrônicas, etc.). Um dos requisitos para esse tipo de software é ter código compacto e de arquitetura neutra, ou seja, uma aplicação que funcione em qualquer plataforma.

3.1.1.1 Plataforma Java

O universo Java é um vasto conjunto de tecnologias, composto por três plataformas principais que foram criadas para segmentos específicos de aplicações (GIONGO; ARAUJO, 2015):

- **Java SE (*Java Platform, Standard Edition*):** É a base da plataforma; inclui o ambiente de execução e as bibliotecas comuns.
- **Java EE (*Java Platform, Enterprise Edition*)**: A edição voltada para o desenvolvimento de aplicações corporativas e para internet.
- **Java ME (*Micro Edition*)**: A edição para o desenvolvimento de aplicações para dispositivos móveis e embarcados.

Além disso, pode-se destacar outras duas plataformas Java mais específicas:

- **Java *Card:*** Voltada para dispositivos embarcados com limitações de processamento e armazenamento.
- **JavaFX:** Plataforma para desenvolvimento de aplicações multimídia para aplicações locais, para Internet (JavaFX *Script*) e dispositivos móveis (JavaFX *Mobile*).

A plataforma Java é constituída de um grande número de tecnologias, cada uma provê uma porção distinta de todo o ambiente de desenvolvimento e execução de software. Os usuários finais, tipicamente, interagem com a máquina virtual JVM (*Java Virtual Machine*) e um conjunto padrão de bibliotecas de classe. Os desenvolvedores de aplicações em Java utilizam um conjunto de ferramentas de desenvolvimento, denominado JDK (*Java Developer Kit*).

3.1.1.2 Características

A plataforma Java implementa o conceito híbrido, que tem como objetivo a segurança das verificações existentes em um processo de compilação e a portabilidade dos ambientes interpretados. O processo adotado para a implementação do modelo híbrido baseia-se na utilização da representação de um modelo intermediário denominado *bytecode*, que é gerada pelo compilador e interpretada no momento da execução (MATTOS, 2007).

Ainda conforme explica Mattos (2007), a plataforma Java utiliza essa

abordagem, contendo os seguintes passos em seu processo de desenvolvimento:

- arquivos com código fonte são armazenados em forma de um arquivo texto comum (.java);
- após a compilação dos arquivos de código fonte são gerados os *bytecodes* (.class);
- os *bytecodes* por fim são interpretados pela Máquina Virtual Java, que por conseguinte transforma-os em linguagem de baixo nível (Linguagem de máquina – binária).

Um recurso fundamental do Java é o "*garbage collector*", que se responsabiliza por liberar parte da memória alocada que não é mais referenciada. Assim, os programadores não necessitam, e nem podem, liberar explicitamente memória anteriormente alocada.

O nome coleta de lixo é associado ao fato de que, se objetos não estão mais sendo utilizados, então pode-se "jogá-los fora". Outra metáfora para esse recurso é a "reciclagem" de memória. Quando um programa não mais referencia um objeto, esta parte da memória pode ser liberada para que outros objetos subsequentes, possam ali serem alocados.

A coleta de lixo alivia os programadores da questão de liberar a memória alocada, uma vez que esse processo é considerado complicado e perigoso para o nível de aplicação. A coleta de lixo é parte importante da estratégia de segurança da plataforma, pois se ela for feita de maneira incorreta pode causar uma falha crítica no funcionamento da JVM, e assim comprometer a aplicação que está sendo executada.

3.1.2 NetBeans IDE

O NetBeans IDE, segundo Oracle (2015), é um IDE *(Integrated Development Environment*), gratuito e de código aberto para desenvolvedores de software para diversas linguagens de programação, inclusive o Java e funciona em diversos

sistemas operacionais. O NetBeans IDE oferece aos desenvolvedores, ferramentas necessárias para criar aplicativos profissionais de *desktop*, empresariais, para redes de computadores e Internet e móveis multiplataformas.

A IDE NetBeans auxilia programadores a escrever, compilar, depurar e instalar aplicações e foi arquitetada na forma de uma estrutura reutilizável que visa simplificar o desenvolvimento e aumentar a produtividade, pois reúne em uma única aplicação diversas funcionalidades.

A IDE NetBeans auxilia programadores a escrever, compilar, depurar e instalar aplicações, e foi arquitetada em forma de uma estrutura reutilizável que visa simplificar o desenvolvimento e aumentar a produtividade, pois reúne em uma única aplicação todas estas funcionalidades.

De acordo com o sítio oficial do projeto (NETBEANS, 2015), o NetBeans foi iniciado em 1996 por dois estudantes tchecos na Universidade de Charles, em Praga, quando a linguagem de programação Java ainda não era tão popular como atualmente. Já em 1999 o projeto já havia evoluído para uma IDE proprietário, com o nome de NetBeans DeveloperX2, nome que veio da ideia de reutilização de componentes que era a base do Java. Nessa época a empresa *Sun Microsystems* havia desistido de sua IDE Java Workshop e, procurando por novas iniciativas, adquiriu o projeto “NetBeans DeveloperX2” incorporando-o a sua linha de softwares.

3.1.3 Android

O Android é uma plataforma para tecnologia móvel, envolvendo um pacote com programas para celulares, já com um sistema operacional *middleware,* aplicativos e interface com o usuário (PEREIRA; SILVA, 2009).

Android foi construído com a intenção de permitir aos desenvolvedores criar aplicações móveis que possam tirar total proveito do que um aparelho portátil possa oferecer. Foi construído para ser verdadeiramente aberto. Por exemplo, uma aplicação pode utilizar qualquer uma das funcionalidades de núcleo do telefone, tais como efetuar chamadas, enviar mensagens de texto ou utilizar a câmera, que permite aos desenvolvedores adaptarem e evoluírem cada vez mais estas

funcionalidades.

O sistema Android, é o primeiro projeto de uma plataforma de código livre para dispositivos móveis. O SDK é o kit de desenvolvimento que disponibiliza as ferramentas e API's necessárias para desenvolver aplicações para a plataforma Android, utilizando a linguagem Java.

A plataforma Android foi desenvolvida com base no sistema operacional Linux, no entanto não é considerado como sendo um Linux, pois não possui sui *windowing system* nativo (componente de interface gráfica), não suporta *glibc* (GNU, 2015) e não possui alguns dos conjuntos de padrões apresentados em algumas distribuições Linux.

O Android é considerado muito seguro. Como ele é executado em um *Kernel* (núcleo) Linux, toda a vez que um aplicativo for instalado no dispositivo, é criado um novo usuário Linux para aquele programa, com diretórios que serão usados pelo aplicativo, mas somente para aquele usuário Linux. Como cada aplicação instalada fica completamente isolada das outras, qualquer tentativa de acessar informações de outro aplicativo precisa ser explicitamente autorizada pelo usuário, podendo ser negada a sua instalação ou autorizada, mas controlando as permissões que este aplicativo poderá ter por meio de um mecanismo de permissões (PEREIRA; SILVA, 2009).

3.1.3.1 Arquitetura Android

Na Figura 2 é possível visualizar a estrutura da arquitetura do Android, com cada uma das camadas disponíveis e funcionalidades (GOOGLE, 2015).

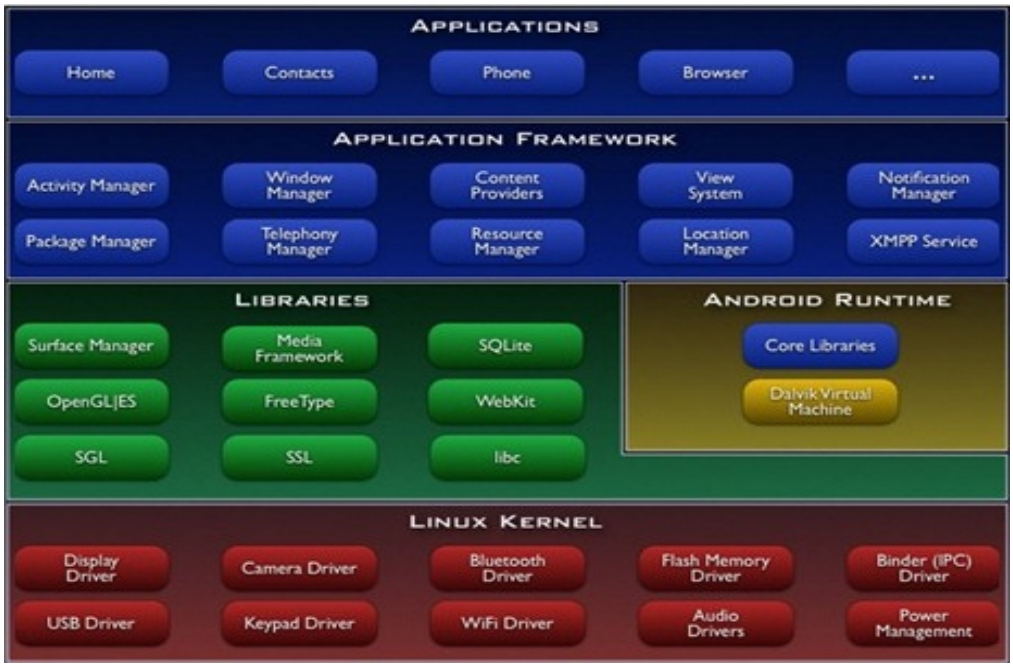

Figura 2 - Arquitetura Android

Fonte: Google (2015).

- **Camada de Aplicações:** na camada de aplicações podem existir todos os aplicativos fundamentais (escritos em Java) do Android, como por exemplo, navegadores web, clientes de e-mail, calendários, mapas, gerenciadores de contatos, aplicações de voz e todas as demais que são em nível de usuário.

- **Camada de *Framework*:** nesta camada são encontradas as Bibliotecas de código fonte e os recursos utilizados pelos aplicativos, como classes visuais (componentes de interface gráfica), gerenciadores de localização (para aplicações de mapa e posicionamento global), gerenciadores de notificações (fornecem informações sobre o que acontece no dispositivo).

- **Camada de Bibliotecas:** carrega consigo um conjunto de bibliotecas C/C++ utilizadas pelo sistema e também bibliotecas multimídia, visualização de camadas 2D e 3D, funções para navegadores para Internet, funções para gráficos e aceleradores de hardware, renderização imagens tridimencionais, funções de acesso a banco de dados, e muitos outros recursos estão disponíveis no *framework* para o desenvolvimento de aplicativos.

- **Camada Android *Runtime*:** é uma instância da máquina virtual *Dalvik*, criada para cada aplicação ser executada no Android. A *Dalvik* é uma máquina virtual onde cada aplicação é executada dentro do sistema Android, e possui bom desempenho, boa integração com a nova geração de hardware e projetada para executar várias máquinas virtuais paralelamente. Foi projetada para funcionar em sistemas com baixa frequência de processamento, pouca memória de trabalho e sem área de troca de memória e além de tudo para ter maior rendimento do consumo da bateria.

- **Linux *Kernel*:** Utiliza a versão 2.6 do *Kernel* (núcleo) do Linux para serviços centrais do sistema, tais como segurança, controle de memória, de processos, protocolos de rede e de modelos de drives. Também se encontra um módulo de software capaz de gerenciar energia, ele identifica quais recursos do aparelho não estão em uso e os desliga.

3.1.4 JSON (*JavaScript Object Notation*)

O JSON (*JavaScript Object Notation*), é um modelo para transmissão e armazenamento de informações em modo texto, utilizado principalmente na Internet para trocar informações entre clientes e servidores. Para os seres humanos, é fácil de ler e escrever, para máquinas, é fácil de interpretar e gerar.

JSON está constituído em duas estruturas (JSON, 2015):

- Uma coleção de pares rótulo-valor. Em várias linguagens, isto é caracterizado como um objeto composto de vários atributos seguido de seus valores

- Uma lista ordenada de valores. Na maioria das linguagens, isto é caracterizado como um vetor, lista ou sequência.

No Quadro 1 é possível visualizar a estrutura de objeto JSON.

```
{
     “titulo”: “JSON”,
     “resumo”: “Resumo do livro x”,
     “ano”: 2012,
     “genero”: [“aventura”, “ação”, “ficção”]
}
```

Quadro 1: Representação de um objeto JSON.

Conforme visto no Quadro 1, o rótulo “titulo” é a chave para representação do atributo título que posteriormente irá compor o objeto depois de trafegado pela rede e montado no local destino.

Para que um objeto JSON possa ser trafegado de um dispositivo de origem até um destino é necessário realizar a sua conversão, também conhecida como *parser,* que consiste em converter o objeto populado para uma *string* e em seguida encaminhar até o dispositivo de destino. No dispositivo de destino é executado o processo inverso: a s*tring* recebida pelo destino deverá novamente ser transformada em um objeto por meio de uma função *parser.*

Uma característica ressaltante do JSON é que ele pode ser usado para trafegar dados entre diferentes tipos de aplicações independendo de sua linguagem utilizada na construção. Isso é possível porque ele *se baseia string* para o tráfego das informações. s*tring* é um tipo de dado amplamente suportado pelas linguagens de programação, tornando o JSON uma ferramenta popular, usada na integração de sistemas computacionais.

3.1.5 O JSF (*Java Server Faces*)

O JSF é uma especificação padrão de um *framework* utilizada na plataforma Java para construção de interfaces visuais de usuário baseadas em componentes para aplicações web. Possui uma metodologia de programação orientada e eventos, dessa forma acaba abstraindo os detalhes da manipulação dos eventos e organização dos componentes, possibilitando o programador a se preocupar com a lógica de negócio e não mais com o desenvolvimento dos componentes de tela.

Segundo Oracle(2015), JSF oferece facilidade de uso das seguintes formas :

- Facilita a construção de uma interface visual usando um conjunto de componentes de interfaces visuais reutilizáveis
- Simplifica a migração de dados da aplicação para a interface visual e provenientes dela
- Ajuda a gerenciar o estado da interface visual nas solicitações do servidor, isto é, a sua sessão bem como os dados contidos nela;
- Oferece um modelo simples para conectar os eventos, como por exemplo, um clique do mouse, gerados pelo cliente ao código da aplicação do servidor
- Permite personalizar os componentes de interfaces visuais para que sejam facilmente construídos e reutilizados.

De forma geral, a utilização do JSF possibilita mais produtividade quando relacionado ao desenvolvimento da aplicação, justamente pelo fato dos componentes de tela estarem prontos. A alta produtividade e um alto grau de abstração são pontos muito fortes da tecnologia.

3.1.5.1 Primefaces

Não há componentes sofisticados dentro da especificação padrão do JSF e isso é proposital: uma especificação tem que ser estável e as possibilidades das

interfaces com o usuário crescem muito rapidamente, a especificação trata do que é fundamental, mas outros projetos como o *Primefaces* suprem o que falta. O *Primefaces* é uma suíte de componentes de código livre, com foco na construção de interfaces de usuários para aplicações Web, e baseada em Java EE (*Java Enterprise Edition*), atualmente está na versão 5.0 com uso da especificação JSF 2.2. Segundo Pessoa (2012), o uso do JSF permite o padrão de projeto MVC[4] (modelo, visualização, controle), desse modo, oferece a clara separação entre a visualização e regras de negócio (modelo) (PRIMEFACES, 2015).

O *Primefaces* tem atraído a preferência de muitos desenvolvedores ao observar alguns pontos interessantes: inserção no projeto utilizando somente um arquivo Jar de aproximadamente 1.7 MB, facilidade de uso, não é necessário que o programador saiba como funcionam os componentes para que ele possa utilizá-los. Possui suporte a vários navegadores, uma comunidade muito colaborativa, sem dependências requeridas, uso difundido, compatibilidade com HMTL 5, vários componentes de formulários.

3.1.6 Maven

Segundo o sítio oficial do projeto (MAVEN, 2015), o *Maven* é uma ferramenta utilizada para gerenciar projetos em Java e simplificar o trabalho do programador, auxiliando no ciclo de desenvolvimento, incluindo compilação, controle de bibliotecas, distribuição e relatórios estatísticos. O projeto nasceu a partir das dificuldades encontradas principalmente em gerenciar a compilação de projetos e no controle de bibliotecas .

O *Maven* é um projeto de código livre, mantido pela *Apache Software Foundation*. Desde seu início, o *Maven* tem sido utilizado por projetos de código livre e também proprietário. Com a versão 2, o Maven passou de uma ferramenta de construção para uma ferramenta complexa de gestão de construção de software,

4 **Model-view-controller** (**MVC**), em português **modelo-visão-controlador**, é uma estratégia de separação de camadas de software que visa desacoplar a interface de seu tratamento e de seu estado (SAMPAIO,2007).

aplicável a maioria dos cenários de desenvolvimento de software.

O *Maven* baixa bibliotecas Java e suas extensões dinamicamente de um ou mais repositórios remotos, como o *Maven* 2 Central *Repository*, e armazena-os em uma área de cache local. Este cache local de artefatos baixados pode também ser atualizado com artefatos criados por projetos locais. Repositórios públicos podem também ser atualizados.

Em resumo, o *Maven*:

- Define como o projeto é tipicamente construído;
- Utiliza convenções para facilitar a configuração do projeto e assim, sua construção;
- Ajuda os usuários a compreender e organizar melhor a complexa estrutura dos projetos e suas variações;
- Prescreve e força a utilização de um sistema de gerenciamento de dependências comprovadamente eficaz, permitindo que times de projeto em locais diferentes compartilhem bibliotecas;
- É flexível para usuários avançados, permitindo que definições globais sejam redefinidas e adaptadas de forma declarativa (alterando-se a configuração, alterando metadados ou através da criação de plug-ins).
- Está em constante evolução, incorporando novas práticas.

Como pode ser visto o *Maven* é muito importante para simplificar o trabalho do programador, principalmente para controlar as dependências de um projeto Java, dessa forma ele está cada vez mais popular na comunidade Java.

3.1.7 GlassFish

O *GlassFish* é um servidor de aplicação para sistemas web de código livre atualmente patrocinado pela Oracle, para a plataforma Java EE, que disponibiliza um ambiente para a instalação e execução de certas aplicações, centralizando e

dispensando a instalação nos computadores clientes. Os servidores de aplicação também são conhecidos por *middleware*.

O *GlassFish* suporta todas as especificações padrões da plataforma Java, portanto todas as implementações como por exemplo o *WebSocket*, já são suportadas nativamente pelo servidor de aplicação, não sendo necessário importar bibliotecas ou extensões para que ela funcione.

4 RESULTADOS E DISCUÇÕES

Este capítulo apresenta as principais características do ramo de negócio para o qual o sistema está sendo desenvolvido.

4.1 Descrição Geral

Nesta Seção é apresentada uma abordagem geral da descrição do cenário de funcionamento do sistema e da solução proposta.

4.1.1 Cenário

Muitas pessoas necessitam que seus dispositivos móveis sejam rastreados, ou mesmo dispositivos de terceiros, para isto é necessário que a sua localização seja capturada de tempos em tempos, isto é, as coordenadas geográficas devem ser registradas em uma base de dados e posteriormente disponibilizadas para que ele próprio ou outro indivíduo possa monitorar a sua movimentação, como por exemplo, tracejando a rota em um mapa. Na figura 3 é apresentada em sequencia as ações do negócio, bem como os passos para o resultado:

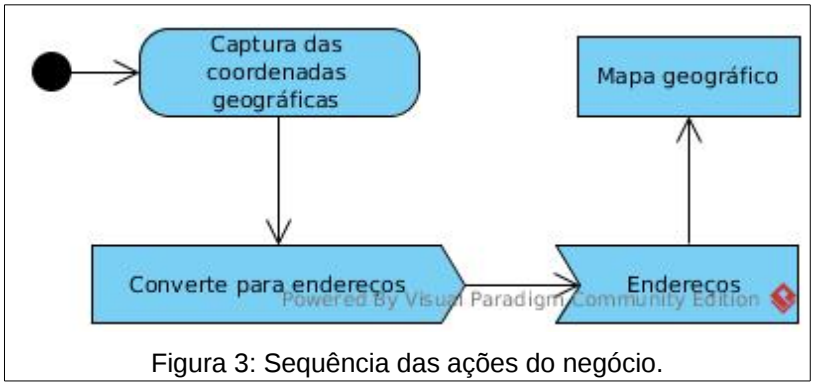

Figura 3: Sequência das ações do negócio.

A primeira ação no momento em que uma pessoa é submetida ao rastreamento de sua localização, é obter as suas coordenadas geográficas. As

coordenadas geográficas definem a posição do indivíduo em um ponto do globo terrestre, a próxima ação, será converter essas coordenadas num endereço para que seja possível conhecer, como por exemplo, o nome da rua ou cidade onde ele se situa. O próximo passo será identificar em um mapa, os pontos exatos, ou seja, identificar por meio das coordenadas geográficas os pontos por onde o indivíduo passou, com seus respectivos endereços para que seja mais amigável a leitura da trajetória.

Por meio dessa sequência apresentada, quando o posicionamento de uma pessoa é registrado a cada certo período de tempo e apresentando num mapa, é possível que seja gerado um histórico da sua localização, permitindo acompanhar de forma precisa cada movimentação que ela sofreu ao longo do tempo.

4.1.2 Solução Proposta

O sistema desenvolvido é composto basicamente por dois módulos:

i. Móvel (cliente): Aplicativo móvel cliente que tem por finalidade capturar, armazenar e transmitir as informações de localização do aparelho;

ii. Aplicação Web (servidor): Aplicativo Web para cadastro e manutenção dos dispositivos móveis, bem como acompanhamento das localizações.

Na Figura 4 são apresentados os módulos do sistema:

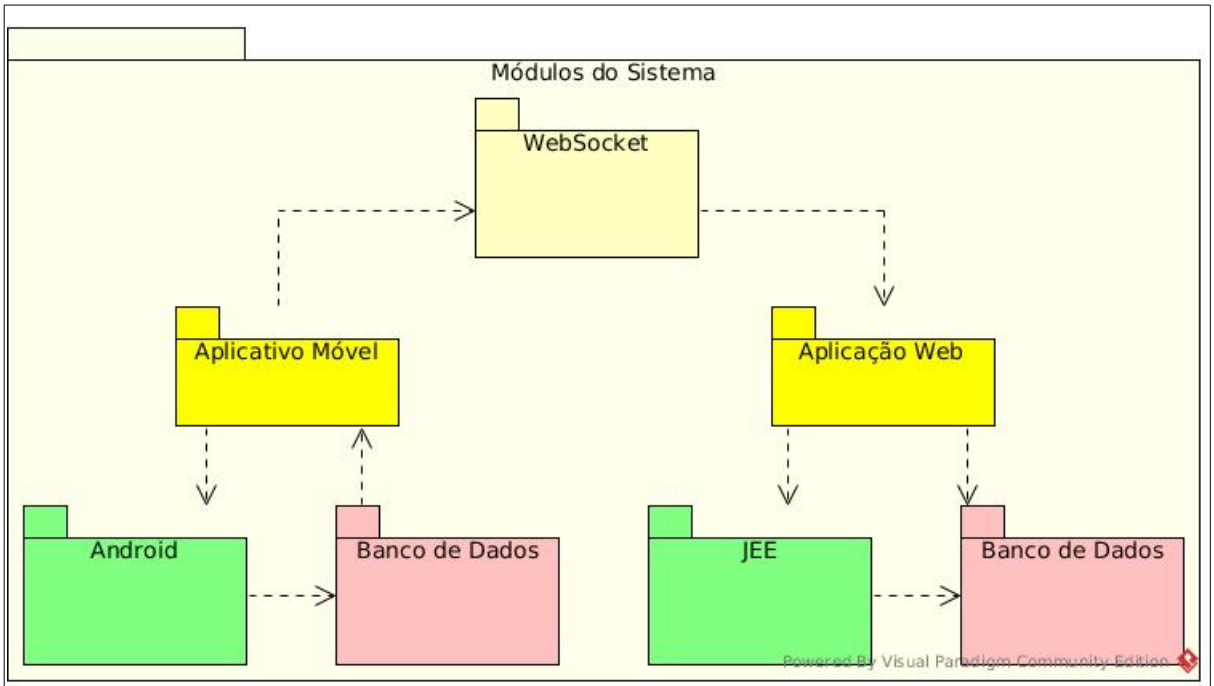

Figura 4: Módulos do sistema.

Conforme pode ser visualizado na Figura 4, o aplicativo móvel tem por finalidade capturar e armazenar as informações das coordenadas geográficas do dispositivo móvel e comunicar-se diretamente com a aplicação web, enviando os dados das coordenadas por meio da implementação de um cliente *WebSocket*. Quando não existe conectividade com a rede, a aplicação móvel persiste diretamente em uma base de dados local, para que posteriormente quando a conectividade estiver disponível, os dados possam ser enviados.

Na aplicação web (Módulo servidor), quando detectado o envio de dados de um cliente, este identifica a sua origem, em seguida os dados são armazenados de forma definitiva na sua base de dados e disponibilizados para que o usuário posteriormente possa consultá-los.

4.2 Especificação e Requisitos do Sistema

As pessoas que usam o sistema são identificadas como atores (que executam um papel no sistema). Como os usuários possuem algumas funções específicas no sistema, a descrição realizada na Quadro 2 identifica, resumidamente quem são estes usuários e no Quadro 3, a vinculação de suas funções.

Nesta seção também está os diagrama de classes e também os documentos da representação da arquitetura do sistema e dos padrões de projeto utilizados.

Nº	Nome do Ator	Descrição
1	Monitorador (A)	Pessoa que pode realizar todas as funções no sistema.
2	Monitorado (B)	Pessoa que utilizará o módulo móvel responsável por obter a localização geográfica.

Quadro 2: Atores e seus papéis.

Conforme visualizado no Quadro 2, os principais atores do sistema são os usuários Monitorador e o usuário Monitorado.

Nº	Nome do Caso de Uso	Atores	
		A	B
1	Manter Usuário	X	
2	Manter Dispositivos	X	
3	Visualizar Históricos de locais	X	
4	Visualizar Gráfico de distâncias	X	
5	Utilizar o módulo móvel	X	X

Quadro 3: Atores e suas funcionalidades.

No Quadro 3, como pode ser visto, o Monitorador possui acesso a todas as funcionalidades, inclusive acessar o cliente móvel.

4.3 Listagem de Atores e Casos de Uso

Para esse sistema foram identificados os seguintes atores:

- **Monitorador:** é o usuário que acesso total ao sistema, em resumo é o usuário que poderá acessar pela primeira vez o sistema e terá a permissão para cadastrar muitos dispositivos e acompanhar as demais informações geradas;

- **Monitorado:** é o indivíduo que será rastreado pelo sistema e que terá acesso ao módulo móvel.

Como funcionalidades do módulo *Web* (servidor) têm-se: controle de acesso, manutenção de dispositivos, visualização de histórico de localizações por dispositivo cadastrado, visualização de gráfico estatístico contendo o total da distância percorrida por cada dispositivo num certo período de tempo. No módulo Móvel (cliente): tem-se uma aplicação que funciona através de um processo que tem por finalidade capturar as posições quando o usuário se movimenta.

As funcionalidades do sistema foram modeladas em casos de usos organizados em diagramas. Esta modelagem permitirá um acompanhamento do requisito nas diversas etapas do processo de desenvolvimento.

Para representar as funcionalidades de manutenções de dados do sistema proposto, o diagrama de casos de uso foi modelado e está representado na Figura 5.

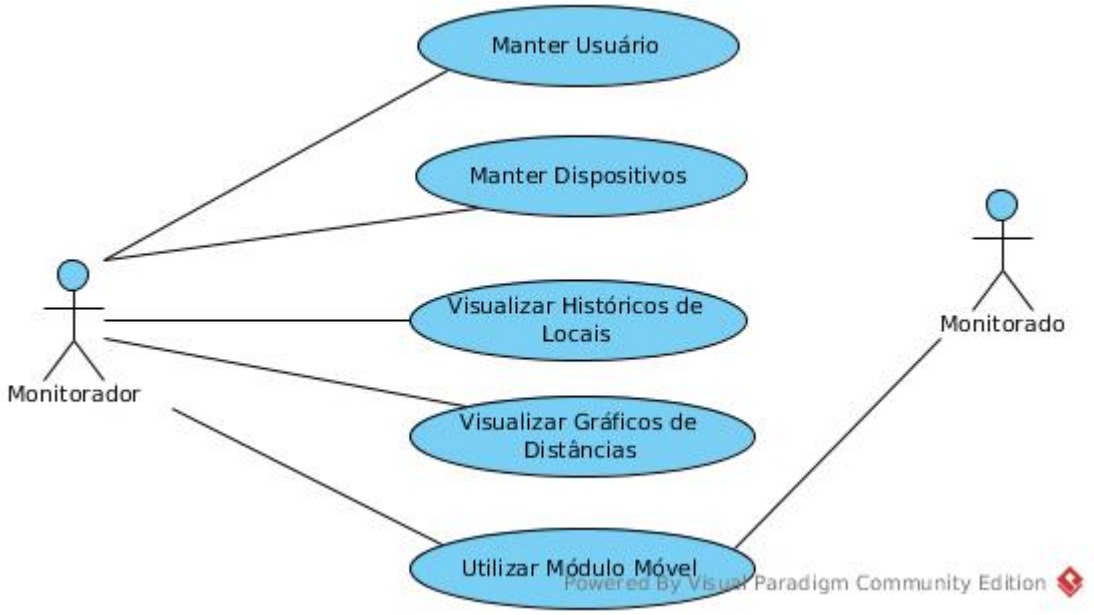

Figura 5: Diagrama de Casos de Uso.

O Monitorador é o ator que possui total acesso às funcionalidades do sistema, visto que ele poderá gerenciar todas as informações relacionadas aos usuários que serão rastreados, bem como visualizar as informações de históricos de localizações e estatísticas de distâncias. O Monitorado será o ator que porta em seu dispositivo móvel, o módulo móvel, que tem por simples finalidade registrar as coordenadas conforme ele se movimenta no decorrer do dia. Claro que nada impede que o Monitorador faça uso do módulo móvel e portanto, ser caracterizado como um usuário Monitorado e registrar a sua própria localização e acompanhar por meio do módulo Web.

4.4 Arquitetura Padrão de Projeto

A arquitetura de projeto utilizada (Figura 6) é o modelo de três camadas MVC (Modelo-Visualização-Controle), por fornecer uma maneira de dividir a funcionalidade envolvida na manutenção e apresentação dos dados da aplicação.

MVC é um modelo de desenvolvimento de Software, atualmente

considerado uma "arquitetura padrão" utilizada na Engenharia de Software. O modelo isola a lógica (a lógica do programa) da interface do usuário (Inserir e exibir dados), permitindo desenvolver, editar e testar separadamente cada parte. Basicamente o padrão MVC implementa um caso de uso interativo em três componentes (SAMPAIO, 2007; p.79):

- **Modelo:** mantém o estado atual do módulo de Visualização. É responsável por alterar o estado e fornecer à Visualização os valores atuais;

- **Visualização:** captura ações do usuário e comunica ao Controle para que sejam executadas. Também atualiza as informações exibidas sempre que o Controle avisa que devem ser atualizadas;

- **Controlador:** recebe solicitações da Visualização, invoca as regras de negócio necessárias, comanda a alteração do estado do Modelo e a atualização das informações exibidas na Visualização.

No sistema desenvolvido, a arquitetura padrão MVC, utilizada é representada conforme a Figura 6.

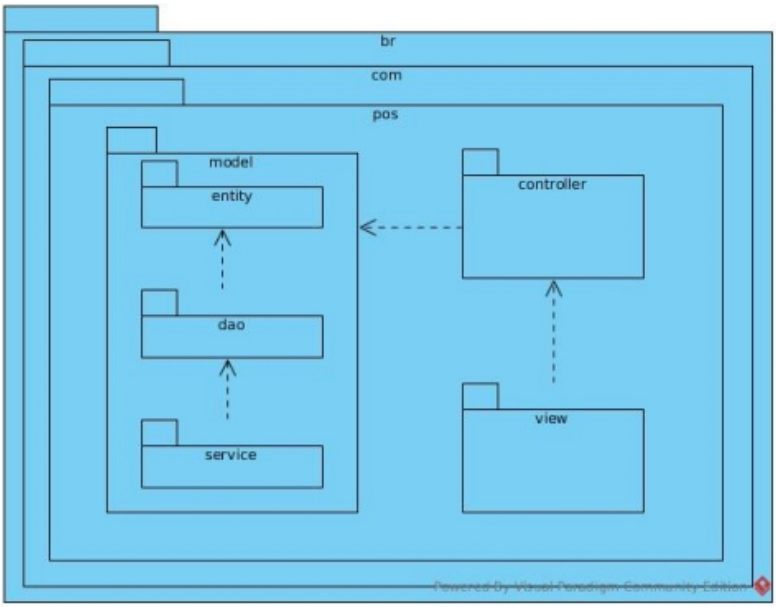

Figura 6: Arquitetura padrão de projeto.

Conforme pode ser visualizado na Figura 6, a camada de Visualização (*View*), é responsável por manter o funcionamento das janelas (interface com o usuário). As informações após serem processadas pela camada de Visualização são enviadas para a camada de Controle (*Controller*), onde é realizado o processamento intermediário, ou seja, a preparação dos dados para serem persistidos ou recuperados da base de dados. Após essa etapa, os dados são enviados ou recuperados da base de dados, por meio da camada de Modelo (*Model*), que é inteiramente responsável por realizar as transações com a base de dados.

4.5 Diagrama de Classes

O diagrama de Classes apresenta as entidades que representam os objetos abstraídos do modelo real, bem como os relacionamentos entre eles, conforme visto na Figura 7.

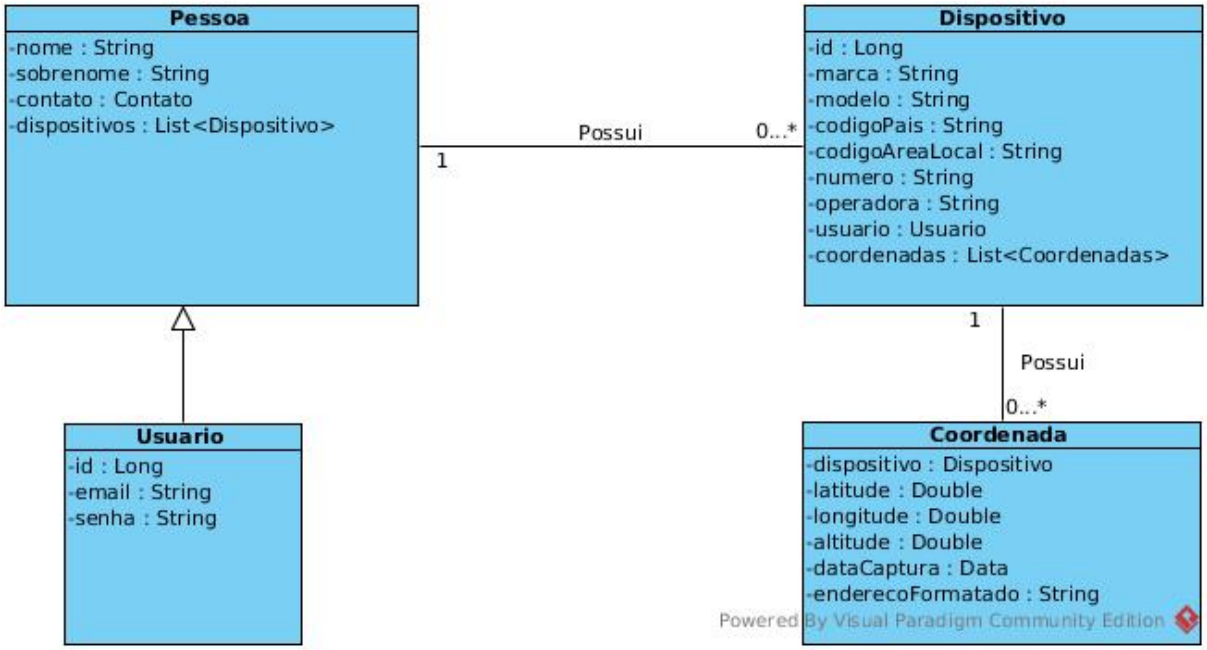

Figura 7: Diagrama de Classes.

Por meio da Figura 7, pode-se visualizar a classe pessoa como sendo a classe com maior importância no sistema, já que ela representa as informações básicas de cada usuário. É possível observar que cada pessoa usuária do sistema estará associada a muitos dispositivos, e estes por sua vez terão uma série de coordenadas, ou seja, os locais registrados pelo GPS do dispositivo móvel são representados pela classe “Coordenada”. Ainda na classe “Coordenada” é possível visualizar o atributo “enderecoFormatado”, que tem por finalidade armazenar no banco de dados, um endereço que seja legível para o usuário, como por exemplo, um endereço que contenha nome da rua, número, bairro etc.

O diagrama também representa a modelagem do banco de dados, uma vez que é utilizada a tecnologia de banco orientado a objetos para persistência de dados na aplicação cliente e de gerenciamento.

4.6 Sistema Desenvolvido e Código Fonte

Nesta Seção são apresentados os resultados obtidos no o desenvolvimento do sistema, incluindo o módulo web e o módulo móvel.

4.6.1 Módulo Móvel

O modulo móvel foi desenvolvido para Android versão 4.1.2 ou superior que representa a maioria dos dispositivos móveis que utilizam esta tecnologia.

O módulo móvel desenvolvido na tecnologia Android, visa contemplar uma única funcionalidade que o sistema necessita neste módulo: capturar os pontos (coordenadas geográficas), de tempos em tempos, a fim de conhecer a localização do dispositivo no decorrer de um período.

A funcionalidade de captura de coordenadas funciona neste módulo como um serviço do sistema operacional, isto é, o usuário do dispositivo, não consegue visualizá-lo por meio de uma tela, pois o processo fica escondido rodando na memória, registrando assim as coordenadas e persistindo no banco de dados local, para que posteriormente sejam enviadas para o módulo web.

4.6.1.1 Permissões

Para acessar recursos do dispositivo é preciso deixar explicito no arquivo “AndoridManifest.xml” as permissões desejadas.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
```

Quadro 4: Arquivo de permissões "AndroidManifest.xml".

Conforme visto do Quadro 4, existem muitas permissões que o aplicativo móvel necessita para que ele funcione corretamente: acesso a Internet, status da rede, gravação de dados na memória, aprimoramentos na localização, estado do

telefone e inicialização automática ao iniciar o sistema operacional. Neste arquivo podem ser informadas todas as permissões que se achar necessário pelo desenvolvedor.

4.6.1.2 Serviço

A captura das coordenadas, depende de um serviço rodando de maneira oculta no sistema, isto é, em forma de um serviço que roda em segundo plano no sistema operacional. Para que esse serviço possa ser criado quando a aplicação móvel é inicializada, uma classe foi codificada para realizar essa tarefa, como pode ser visto no Quadro 5:

```
public class LocalizacaoService extends Service {
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        super.onStartCommand(intent, flags, startId);
        addLocationListener(this);
        Toast.makeText(this, "Serviço de localização iniciado...",
Toast.LENGTH_LONG).show();
        return START_STICKY;
    }
    @Override
    public void onDestroy() {
        super.onDestroy();
        Toast.makeText(this, "O serviço de localização foi parado...",
Toast.LENGTH_LONG).show();
    }
    private void addLocationListener(final Context contexto){
        Thread triggerService = new Thread(new Runnable(){
            public void run(){
                Looper.prepare();
                LocationManager locationManager =
(LocationManager)getSystemService(Context.LOCATION_SERVICE);
                Criteria c = new Criteria();
                c.setAccuracy(Criteria.ACCURACY_COARSE);
                final String PROVIDER = locationManager.getBestProvider(c, true);
                LocalizacaoListener localizacaoListener = new
LocalizacaoListener(contexto);

locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 2000, 100,
localizacaoListener);
                Looper.loop();
            }
        }, "LocationThread");
        triggerService.start();
   }
}
```

Quadro 5: Classe do serviço do GPS.

A classe “*LocalizacaoService*”, que pode ser vista no Quadro 5, implementa os métodos abstratos da classe “*Service*”. O método “*onStartCommand*” é invocado toda a vez que a classe é instanciada, ou seja, no momento em que a aplicação é aberta e por sua vez chama outro método chamado “*addLocationListener*” que por fim instancia outro objeto responsável por capturar as coordenadas geográficas. A classe que captura as coordenadas é apresentada na Seção a seguir.

4.6.1.3 Captura das Coordenadas

A captura das coordenadas geográficas acontece por meio de uma classe chamada “*LocalizacaoListenner*”, que nada mais é do que uma classe que implementa a interface “*LocationListenner*” e um dos seus principais métodos, e o “*onLocationChanged*”, que nada mais é do que um método ouvinte, e toda a vez que o GPS do dispositivo capturar informações de coordenadas ele é invocado, passando por parâmetro um objeto com o resultado das coordenadas geográficas.

```
public class LocalizacaoListener implements LocationListener {
    public LocalizacaoListener(Context context) {
    }
    @Override
    public void onLocationChanged(Location location) {
        Calendar calendar = Calendar.getInstance();
        LocalHistorico localHistorico = new LocalHistorico();
        localHistorico.setLatitude(location.getLatitude());
        localHistorico.setLongitude(location.getLongitude());
        localHistorico.setAltitude(location.getAltitude());
        localHistorico.setDataCaptura(calendar.getTime());
        enviaArmazenaCoordenadas(localHistorico);
   }
}
```

Quadro 6: Classe "LocalizacaoListenner".

Conforme visto no Quadro 6, quando o método “onLocationChanged” é invocado ao capturar uma coordenada geográfica, imediatamente ele atribui a um objeto as informações para que posteriormente possam ser armazenadas na base de dados ou enviadas para o servidor (Ver Seção 4.6.2).

4.6.2 Módulo Web (Servidor)

No módulo web é possível realizar a manutenção dos dados do sistema, ou seja, é por meio dele que o usuário administrador (Monitorador) terá todo o controle dos dispositivos rastreados, bem como manter informações sobre eles. Nas sessões a seguir são mostradas as principais telas e funcionalidades deste módulo.

4.6.2.1 Cadastro de Conta

Este formulário de cadastro, permite que qualquer pessoa que deseja utilizar o sistema, possa informar seu dados básicos e criar sua conta de usuário para que consiga ter acesso ao sistema.

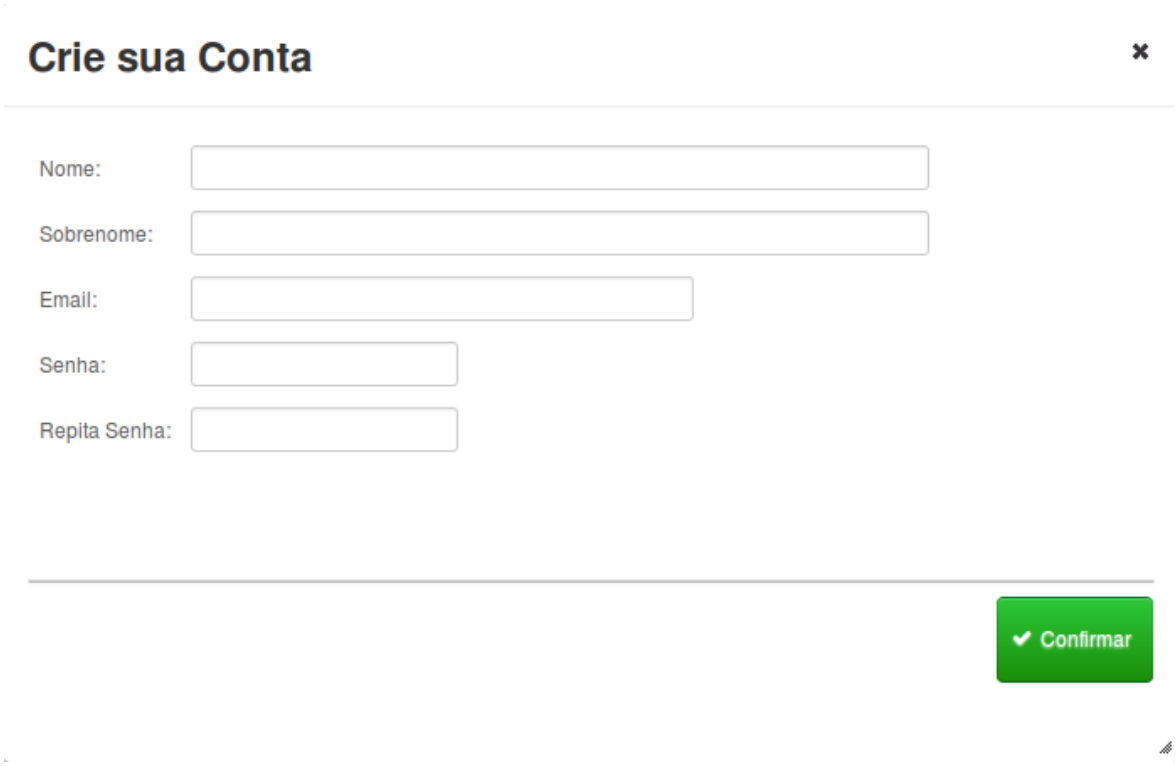

Figura 8: *Dialog* para criação de conta de usuário.

Para a montagem da tela de cadastro de conta foi utilizado um componente do tipo “*Modal Dialog*” (Figura 8), isto é, um componente que flutua sobre as demais telas focando a atenção do usuário para os campos de preenchimento. No Quadro 7 é apresentado o código fonte do arquivo “formulario.xhtml”, que gera os campos na tela:

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core"
      xmlns:p="http://primefaces.org/ui">

    <p:messages id="messages" autoUpdate="true" closable="true"/>
    <h:panelGrid id="grid" columns="2" cellpadding="5">
        <h:outputLabel value="Nome:" for="inputNome" class="ui-widget"/>
        <p:inputText id="inputNome" value="#{usuarioBean.usuario.nome}"
size="60"                 maxlength="100" tabindex="1" required="true"
requiredMessage="Nome           Obrigatório"/>

        <h:outputLabel value="Sobrenome:" for="inputSobrenome" class="ui-
widget"/>
        <p:inputText id="inputSobrenome"
value="#{usuarioBean.usuario.sobrenome}" size="60"  maxlength="100"
       tabindex="2"  required="true" requiredMessage="Sobrenome Obrigatório"/>

        <h:outputLabel value="Email:" for="inputEmail" class="ui-widget"/>
        <p:inputText id="inputEmail" value="#{usuarioBean.usuario.login.email}"
size="40"          maxlength="100" tabindex="3"required="true"
requiredMessage="Email Obrigatório">
        </p:inputText>
        <h:outputLabel value="Senha:" for="inputSenha" class="ui-widget"/>
        <p:password id="inputSenha" value="#{usuarioBean.usuario.login.senha}"
             tabindex="20" redisplay="true" required="true"
requiredMessage="Senha Obrigatória"    />
        <h:outputLabel value="Repita Senha:" for="inputRepitaSenha" class="ui-
widget"/>
        <p:password id="inputRepitaSenha"
value="#{usuarioBean.usuario.login.senha}"    tabindex="20" redisplay="true"
required="true" requiredMessage="Repita Senha              Obrigatório"/>
    </h:panelGrid>
</html>
```

Quadro 7: Código do arquivo "formulario.xhtml"

4.6.2.2 Alteração de Conta de Usuário

Esta tela de manutenção de informações, se refere a alteração dos dados cadastrados descritos na Seção 4.6.2.1 e demonstrada na Figura 9:

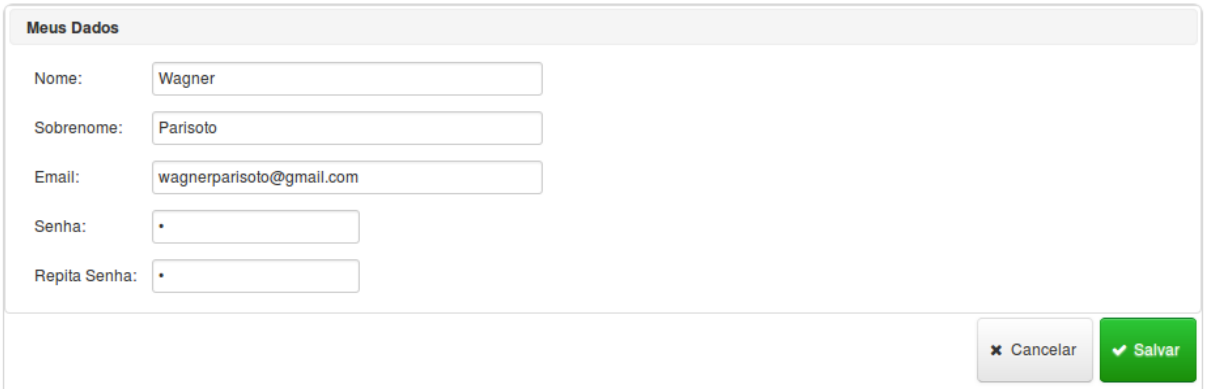

Figura 9: Formulário de alteração de conta de usuário.

Ela permite que o usuário do sistema, neste caso o usuário monitorador dos dispositivos, altere suas informações pessoais informadas quando ele criou sua conta no sistema.

4.6.2.3 Manutenção de Dispositivos

O sistema permite cadastrar muito dispositivos para que sejam rastreados, como pode ser visto na Figura 10, existe um formulário para que sejam informados os dados básicos do dispositivo, como a marca, modelo e o principal deles: o número de telefone, que é utilizado para identificar de forma única o aparelho.

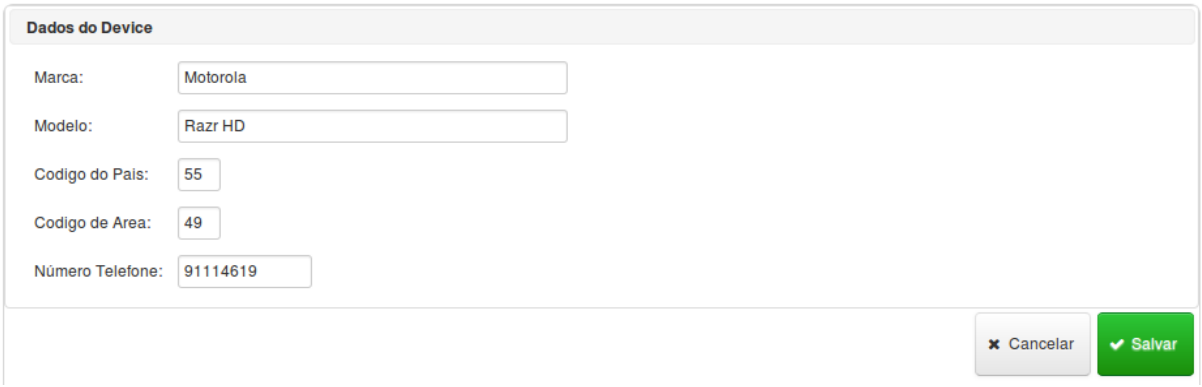

Figura 10: Formulário de dados do Dispositivo móvel.

Existe também uma listagem com os dispositivos cadastrados, com botões para abrir a tela de alteração dos dados inseridos no formulário anteriormente citado e para realizar a exclusão dele no sistema.

Figura 11: Listagem de dispositivos cadastrados.

Conforme visto na Figura 11 os aparelhos são exibidos em forma de "*Card*", e suas informações respectivamente.

4.6.2.4 Histórico e Visualização de Locais

A tela de Histórico e Visualizações de locais, pode ser considerada a mais importante do sistema, já que ela permite de forma muito simples, que o usuário possa visualizar os locais de forma cronológica para o dispositivo selecionado.

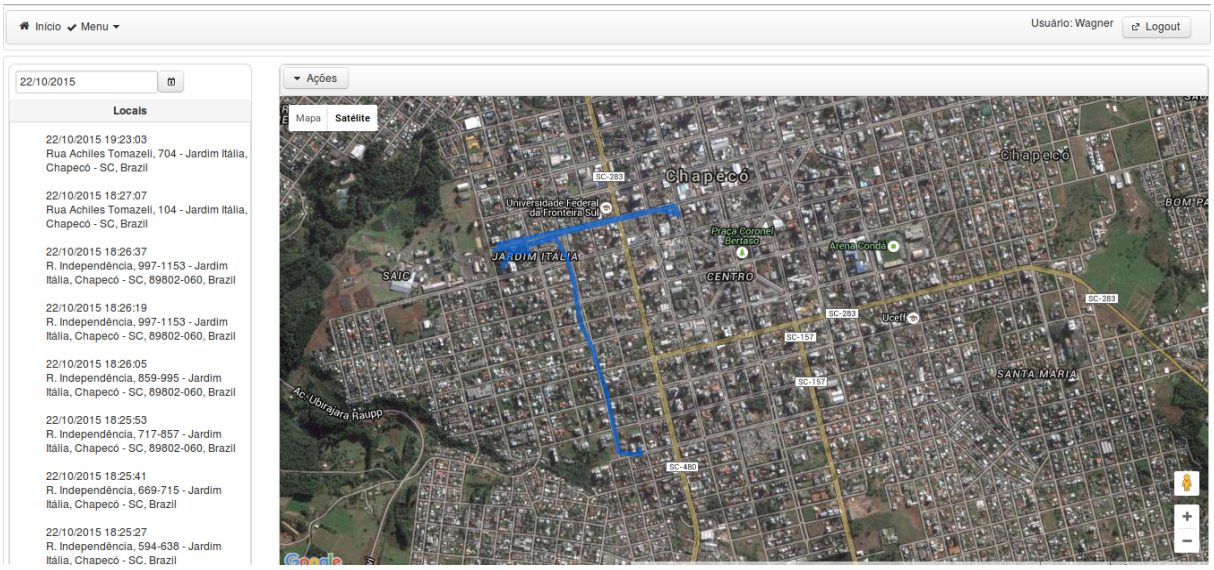

Figura 12: Histórico de locais.

Conforme visto na Figura 12, na caixa lateral esquerda existe uma lista que apresenta todas as coordenadas de forma cronológica, por ordem descendente, isto é, são exibidos, sempre os últimos locais visitados, apresentando a data e hora e o endereço completo do local, para que o usuário tenha uma melhor compreensão. No campo de data acima da caixa, os locais podem ser filtrados por data, exibindo sempre os locais do dia selecionado.

4.6.2.5 Gráfico de Distâncias

Para que o usuário responsável por monitorar os dispositivos cadastrados, possa ter uma noção das distâncias totais percorridas por cada dispositivo, existe no sistema um gráfico que permite comparar as distâncias totais num certo período de tempo.

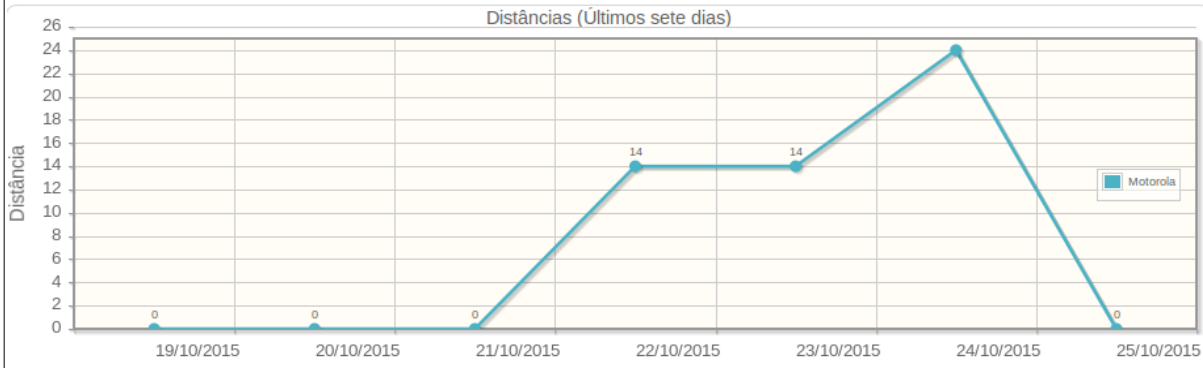

Figura 13: Gráfico de distâncias percorridas.

O gráfico (Figura 13) permite comparar todos os dispositivos com relação a distância total que cada um percorreu nos últimos sete dias corridos, dessa forma é possível ter uma visão estatística com relação a tempo e distância.

```
lineChartModel = initCategoryModel(listaDtos);
        lineChartModel.setTitle("Distâncias (Últimos sete dias)");
        lineChartModel.setLegendPosition("e");
        lineChartModel.setShowPointLabels(true);
        lineChartModel.setAnimate(true);
        lineChartModel.setShowDatatip(true);
        lineChartModel.setLegendRows(2);
        lineChartModel.getAxes().put(AxisType.X, new CategoryAxis("Data"));
        Axis yAxis = lineChartModel.getAxis(AxisType.Y);
        yAxis.setLabel("Distância (Km)");
        yAxis.setTickInterval("3");
        yAxis.setMin(0);
        yAxis.setMax(getDistanciaMaxima(listaDtos));
```

Quadro 8: Código de propriedades do gráfico de distâncias.

No Quadro 8 são demonstradas as propriedades básicas que, programaticamente podem ser definidas, como por exemplo, intervalos de dados de cada eixo, valor mínimo e máximo, incluindo muitos outros.

4.6.2.3 Sincronização

Na Figura 14 pode ser visualizado o fluxo da comunicação do cliente (Módulo móvel) com o servidor (Módulo Web), obedecendo o protocolo implementado para que as mensagens sejam transferidas através da rede. Esse procedimento permite que o dispositivo móvel envie as coordenadas geográficas para o servidor por meio de uma conexão do cliente diretamente com o servidor.

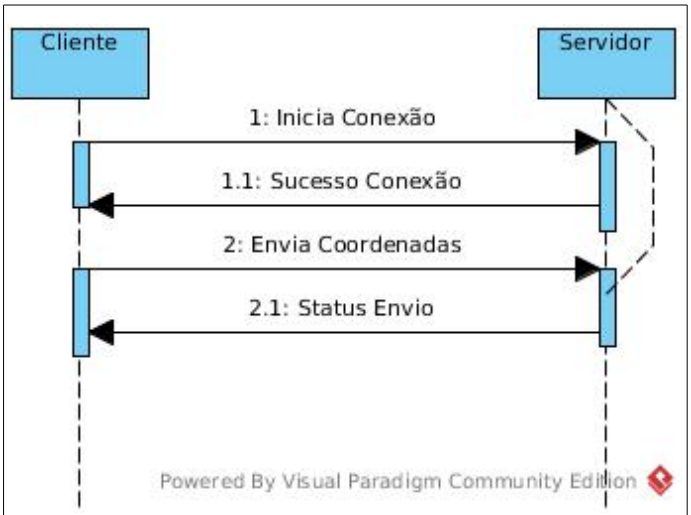

Figura 14: Sequência da comunicação cliente-servidor.

Como pode ser visualizado na Figura 14, o cliente iniciará a conexão com o servidor (Passo 1), estabelecendo as devidas propriedades de comunicação que o *WebSocket* implementa internamente de forma automática. Na sequência (Passo 1.1), o cliente tomará conhecimento da disponibilidade da conectividade e estará ciente que os dados podem ser trafegados por meio do canal. No passo 2, o cliente enviará os dados para o servidor e na sequência (Passo 2.1), receberá por meio de uma mensagem, a resposta contendo o status da transmissão dos dados (Recebido com sucesso).

Para que o dispositivo cliente envie as coordenadas geográficas para o servidor, é utilizado basicamente um Objeto que é valorizado com as coordenadas, convertido para um *hash JSON,* e enviado por meio do cliente *WebSocket* até o

servidor. A Figura 15 representa a estrutura das classes dos objetos:

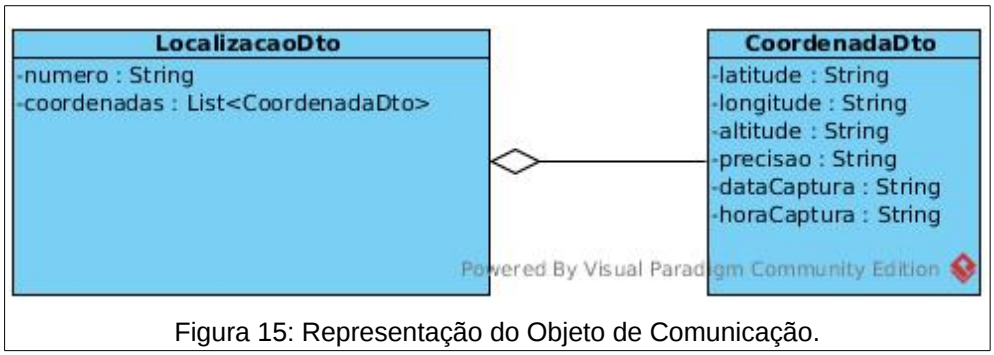

Figura 15: Representação do Objeto de Comunicação.

Como pode ser visto na Figura 15, um objeto do tipo “*LocalizacaoDto*”, possui basicamente dois atributos, um número que se refere ao número da linha móvel, que também é um identificador único para que o servidor reconheça o aparelho. No atributo seguinte pode-se ver uma lista de “CoordenadasDto”, que são as coordenadas geográficas capturadas pelo dispositivo, contendo vários atributos, dentre eles a latitude e longitude, que identificam a posição do dispositivo num determinado momento.

Toda a vez que o módulo móvel possuir conexão com a rede de dados, ele envia as informações de coordenadas capturadas para o servidor. Esse processo ocorre por meio da implementação de um *WebSocket*, que fica aguardando as requisições oriundas dos dispositivos móveis, a fim de receber os dados dos clientes.

A criação de um *WebSocket* em Java é muito simples, basta anotar uma classe qualquer com a anotação “*@ServerEndpoint*”, e passar por parâmetro o endereço URL, que deverá ser invocado pelo cliente, e a classe passa a se comportar como um servidor de requisições capaz de trocar informações entre cliente-servidor. No Quadro 9 pode ser vista a classe que implementa um servidor *WebSocket* no sistema em questão e que é capaz de receber os dados oriundos de clientes:

```
@ServerEndpoint("/localizacao_"WebSocket"")
public class Localizacao"WebSocket" implements Serializable {
    @OnOpen
    public void onOpen(Session cliente) {
        System.out.println("Novo cliente");
    }
    @OnClose
    public void onClose(Session cliente) {
        System.out.println("Cliente desconectado" + cliente);
    }
    @OnMessage
    public void message(String gsonMessage, Session client) throws
IOException {
        Gson gson = new Gson(); ias. 49
        LocalizacaoDto dto = gson.fromJson(gsonMessage,
LocalizacaoDto.class);
        for(CoordenadaDto coordenada : dto.getCoordenadasDto()){
            System.out.println("Latitude: " + coordenada.getLatitude());
            System.out.println("Longitude: " +
coordenada.getLongitude());
            System.out.println("Altitude: " + coordenada.getAltitude());
            System.out.println("Data: " + coordenada.getDataCaptura());
            System.out.println("Hora: " + coordenada.getHoraCaptura());
            System.out.println("Precisao: " + coordenada.getPrecisao());
            System.out.println("=======================\n");
        }
        CallbackDto callback = sincronizaDispositivo(dto);
        Set<Session> openSessions = client.getOpenSessions();
        for (Session session : openSessions) {
            session.getBasicRemote().sendText(gson.toJson(callback));
        }
    }
   private CallbackDto sincronizaDispositivo(LocalizacaoDto
localizacaoDto) {
        DispositivoService dispositivoService = new DispositivoService();
        if (dispositivoService.persitirCoordenadas(localizacaoDto)) {
            return new
CallbackDto(StatusHandshakeEnum.SUCESSO_TRANSFERENCIA);
        }
        return new CallbackDto(StatusHandshakeEnum.FALHA_LOGIN);

    }
```

Quadro 9: Classe *WebSocket* no módulo servidor.

Como pode ser visto no Quadro 9, quando um cliente invocar a URL do sistema e concatenando ao final “/localizacao_WebSocket”, o cliente poderá trocar os dados contendo as coordenadas com o servidor, incluindo uma lista de localizações e no final o cliente receberá uma resposta de retorno que se refere ao status da transmissão dos dados.

O módulo cliente (Módulo móvel), também implementa um *WebSocket*, porém como sendo um cliente, que se conecta com o servidor enviando dados de coordenadas e recebendo mensagens de retorno (Quadro 10).

```
public class "WebSocket" extends Endpoint{
    public Session userSession = null;
    private MessageHandler messageHandler;
    private ClientManager clientManager = null;
    public "WebSocket"(URI endpointURI) throws IOException,
DeploymentException {
        clientManager = ClientManager.createClient();
        clientManager.connectToServer(this,
ClientEndpointConfig.Builder.create().build() , endpointURI);
    }
    @Override
    public void onOpen(Session userSession, EndpointConfig
endpointConfig) {
        this.userSession = userSession;
    }
    @OnClose
    public void onClose(Session userSession, CloseReason reason) {
        this.userSession = null;
    }
    @OnMessage
    public void onMessage(String message) {
        if (this.messageHandler != null) {
            this.messageHandler.handleMessage(message);
        }
    }
```

Quadro 10: Classe "WebSocket" cliente.

Como pode ser visto no Quadro 10, uma classe chamada *WebSocket*, implementa os métodos da Interface “*Endpoint*” e basicamente no seu método construtor são instanciados os objetos necessários para se conectar com o servidor. Também pode ser visto um método “*onMessage*”, onde toda vez que o servidor enviar uma mensagem de retorno, ele será invocado, recebendo por parâmetro a resposta. Quando se deseja enviar uma mensagem para o servidor, é possível por meio de um objeto instanciado da classe *WebSocket*, invocar o método “*sendMessage()*”, passando a mensagem do tipo JSON.

```
final "WebSocket" clientEndPoint = new "WebSocket"(new
URI("ws://192.168.25.6:8080/webapp/localizacao_"WebSocket""));
clientEndPoint.getUserSession().addMessageHandler(new
MessageHandler.Whole<String>() {
    @Override
    public void onMessage(String message) {
        builder.append(“mensagem json”);
    }
});
clientEndPoint.sendMessage(gsonMessage);
clientEndPoint.getUserSession().close();
```

Quadro 11: Enviando dados pelo "WebSocket" cliente.

No sistema em questão, as coordenadas são enviadas por meio da chamada ao método “*sendMessage*()”, do objeto “*clientEndpoint*”, conforme visto no Quadro 11. Logo após o envio da mensagem, o método “*onMessage*()”, da classe “*LocalizacaoWebSocket*”, receberá a mensagem, de acordo com a rotina de código que é executada, conforme apresentada no Quadro 10, visto anteriormente.

Após as coordenadas chegarem no servidor elas são convertidas para endereços reais e persistidas numa base de dados, que posteriormente estarão disponíveis para visualização nas telas do sistema (Ver Seção 4.6.2), como por exemplo na tela de Históricos de Locais, onde são apresentados num mapa por ordem cronológica.

5 CONCLUSÃO

Este trabalho teve como objetivo, apresentar uma solução, de forma simplificada e eficiente para a questão do rastreamento de dispositivos móveis, uma tarefa que a maioria dos proprietários de dispositivos necessitam atualmente. No decorrer do desenvolvimento, foi mantido o foco em diversos quesitos como, elaborar uma solução em que os usuários pudessem rastrear de forma rápida, com baixo custo de processamento por meio de uma aplicação muito leve, tendo prioridade a boa usabilidade do sistema, eficiente disponibilidade das informações e utilizando tecnologias atuais e consolidadas no mercado para seu desenvolvimento.

A partir do sistema desenvolvido, a tarefa de localizar um dispositivo, se tornou muito mais simples já que todas as coordenadas de cada aparelho são concentradas numa tela de fácil visualização e que permite o usuário visualizar as informações de acordo com a data escolhida facilitando a busca e apresentação dos locais num mapa de forma muito intuitiva.

Com o desenvolvimento do trabalho, houve também a necessidade de aplicar muitas tecnologias relacionadas a plataforma Java. Muitas foram vistas no decorrer da especialização e outras não, mas estudadas durante desse trabalho, fazendo com que um nível de conhecimento maior fosse obtido agregando ainda mais para a formação.

6 REFERÊNCIAS

ANATEL. **Pesquisa de Qualidade da Telefonia Móvel no Brasil**. Disponível em: <http://www.anatel.gov.br/Portal/verificaDocumentos/documento.asp?numeroPublicacao=306702&assuntoPublicacao>. Acesso em 22 de Outubro de 2015.

COMER, D. E.; DROMS, R. E. **Redes de Computadores e Internet** / Douglas Earl Comer, Ralph Droms; Porto Alegre – RS: Artmed, 2007.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. **Sistemas Distribuídos: conceitos e projeto** / George Coulouris, Jean Dollimore, Tim Kindberg ; tradução João Tortello. - 5 ed. - Porto Alegre : Bookman, 2013.

ELMASRI, R.; NAVATHE, S. B. **Sistema de Banco de Dados** / Ramez Elmasri, Sham Navathe. São Paulo: Pearson Addison Wesley. 2005.

FERRARI, Fabrício A. **Crie Banco de Dados em MySQL** / Fabrício Augusto Ferrari. São Paulo - SP: Digerati Books, 2007.

FETTE, I; Melnikov, A. **The WebSocket Protocol RFC 6455** / Ian Fette, Alexey Melnikov: Google Inc, 2011.

FONTANA, S. **Sistema de Posicionamento Global: GPS – A Navegação do Futuro**. Porto Alegre: Mercado Aberto, 2002.

GNU, projeto, **The GNU C Library (glibc).** Disponível em: <www.gnu.org/software/libc/>. Acesso em 5 de Novembro de 2015.

GOOGLE, Developers. **Página oficial de desenvolvedores Google**. Disponível em <https://developers.google.com/?hl=pt-br>. Acesso em 16 Outubro 2015.

IDC. **International Data Corporation**. Disponível em: <http://br.idclatin.com/releases/news.aspx?id=1613>. Acesso em 22 de Setembro de 2015.

JSON. 2015. **Página Oficial do Projeto.** Disponível em: <http://www.json.org/> Acesso em: 5 de out. de 2015.

KHOSHAFIAN, Setrag. **Banco de Dados Orientado a Objetos** / Setrag Khoshafian; Rio de Janeiro - RJ: Infobooks, 1999.

LONGLEY, P. A.; GOODCHILD, M. F.; MAGUIRE, D. J.; RHIND, D. W. **Sistemas e Ciência da Informação Geográfica** / Paul A. Longley, Michael F. Goodchild, David J. Maguire, David W. Rhind. Bookman Editora Ltda, Porto Alegre, 2011.

MATTOS, Érico C. T. de. **Programação de Software em Java /** Érico Casella Tavares de Mattos; São Paulo: Digerati Books, 2007.

MAVEN. 2015. **Página oficial do Porjeto**. Disponível em: <https://maven.apache.org/> Acesso em: 5 de out. de 2015.

NASSU, E. A.; SETZER, V. W. **Bancos de Dados Orientados a Objetos** / Eugenio Akihiro Nassu, Valdemar Waingort Setzer; São Paulo - SP: Edegard Blücher, 1999.

NETBEANS. **Welcome to the NetBeans Community.** Disponível em: <www.netbeans.org/about/index.html>. Acesso em 7 de Novembro de 2015.

ORACLE. **Introdução ao JavaServer Faces 2.x**. Disponível em: <https://netbeans.org/kb/docs/web/jsf20-intro_pt_BR.html>. Acesso em 28 de setembro de 2015.

GIONGO, S.; ARAÚJO, C. E. **Introdução as Plataformas Java** / Sheila Giongo, Everton Coimbra de Araújo. Disponível em:

<http://www.devmedia.com.br/introducao-as-plataformas-java/29544>. Acesso em 5 de Novembro de 2015.

PEREIRA, L. C.; SILVA, M. L. **Android para Desenvolvedores** / Lúcio Camilo Pereira, Michel Lourenço da Silva. Rio de Janeiro - RJ: Brasport, 2009.

PESSOA, A. **Java server faces - série frameworks Java.** 2012. Disponível em: <http://www.ameliapessoa.com/2012/01/java-server-faces-serie-frameworks-java.html> Acesso em: 25 set. 2015.

PRIMEFACES. **Página oficial do projeto**. Disponível em: <http://primefaces.org> Acesso em: 30 set. 2015.

SAMPAIO, C. **Guia do Java: Enterprise Edition 5: Desenvolvendo aplicações corporativas** / Cleuton Sampaio. Rio de Janeiro: Brasport, 2007.

TANENBAUM, A. S; STEEN, M. V. **Sistemas Distribuídos: Princípios e Paradigmas** / Andrew Stuart Tanenbaum, Maarten Van Steen. 2 ed.; Pearson, 2006.

UIT. 2015. **Número de aparelhos no mundo**. Disponível em: <http://www.ebc.com.br/tecnologia/2015/05/uit-diz-que-numero-de-celulares-no-mundo-passou-dos-7-bilhoes-em-2015/> Acesso em: 22 de Outubro de 2015.