Revista de Ciências Exatas e Tecnologia

Vol. IV, Nº.4, 5, Ano 2011

LOCALIZAÇÃO VIA SATELITE DESENVOLVIDO PARA DISPOSITIVOS MÓVEIS UTILIZANDO A PLATAFORMA ANDROID

Aqui pode ser colocado um Subtítulo, caso seja a vontade do autor

Ricardo L. R. Freitas Jr
afiliação autor
contato@ricardofreitasjunior.com.br

coautor1
afiliação coautor1
emailcoautor1@dominio

RESUMO

A proposta de desenvolvimento deste protótipo é demonstrar a facilidade proporcionada pela tecnologia podendo ser utilizada como informativo sobre condições meteorológicas, tráfego, guia para rotas, obtenção de maior controle de funcionários e veículos de uma empresa, tais como percurso, horários, velocidade, para dispositivos móveis. O objetivo é desenvolver um protótipo de sistema de localização utilizando o Sistema de Posicionamento Global, direcionado para dispositivos móveis equipados com terminais GPS utilizando a plataforma Android, onde o dispositivo móvel irá receber os dados de satélites GPS, calcular a posição atual do dispositivo, exibir a posição do usuário em um mapa, localizar no banco de dados os pontos próximos exibindo-os no mapa e mostrar uma tela com informações do ponto selecionado, possibilitando também o recalculo do melhor escolha de combustível para abastecer.

Palavras-Chave: Java, Android, GPS, combustível, banco de dados, localização, posicionamento global.

ABSTRACT

The proposed development of this prototype is to demonstrate the ease provided by technology can be used as information about weather, traffic, guide to routes, and better control of staff and vehicles from a company such as route, time, speed, devices furniture. The goal is to develop a prototype tracking system using the Global Positioning System, intended for mobile terminals equipped with GPS using the Android platform, where the mobile device will receive data from GPS satellites to calculate the current position of the device, display the user's position on a map, locate the database points near displaying them on the map and show a screen with information from the selected point, allowing also the recalculation of the best choice of fuel supply.

Keywords: Java, Android, GPS, Database, location, fuel, global positioning.

Anhanguera Educacional S.A.

Correspondência/Contato
Alameda Maria Tereza, 2000
Valinhos, São Paulo
CEP 13.278-181
rc.ipade@unianhanguera.edu.br

Coordenação
Instituto de Pesquisas Aplicadas e Desenvolvimento Educacional - IPADE

Artigo Original / Informe Técnico / Resenha
Recebido em: 30/12/1899
Avaliado em: 30/12/1899

Publicação: 22 de setembro de 2009

1. INTRODUÇÃO

Com o aumento da concorrência entre as empresas e um mercado consumidor cada vez mais exigente, é necessário maior agilidade para solucionar problemas. Devido a esta alta competitividade, surgiram novas tecnologias, no intuito de solucionar problemas com mais eficiência e rapidez estes problemas.

O acesso a informações por meio de uma conexão sem fio é uma das maiores evoluções da tecnologia, sendo permitida pela criação de vários dispositivos móveis. Segundo a INFO (2009): "O mercado de software para celulares é promissor, mas ainda restam muitas questões a responder. Uma das mais importantes é sobre a variedade de sistemas existentes. O ano de 2009 será o ano para os desenvolvedores de aplicativos para sistemas operacionais de dispositivos móveis. 2008 foi o ano da mobilidade, especialmente para quem possui um smartphone." Dentre as novas tecnologias, está a voltada para localização, conhecida como GPS (Global Position System), desenvolvida para dispositivos móveis (PDAs e celulares).

O Sistema Global de Posicionamento, mais conhecido como GPS (Global Positioning Sytem), foi desenvolvido pelo Departamento de Defesa dos Estados Unidas da América, com o propósito militar (Ciência Viva, 2009). Consiste em um conjunto de 24 satélites que enviam sinais de rádio de baixa frequência para um terminal GPS, que mede o tempo de transmissão dos sinais, permitindo calcular a sua localização, utilizando o padrão de sistema de coordenadas internacional WGS-84 e exibindo suas coordenadas através da sua latitude, longitude e altitude, com uma precisão de aproximadamente cinco metros. Esta precisão poder ser afetada por ajustes feitos nos satélites GPS ou por causa da geometria inadequada do satélite. Para que o processo de localização seja efetuado, o dispositivo GPS necessita estar sincronizado com pelo menos quatro destes satélites GPS (NOKIA N95 8GB, 2008).

A linguagem Java é orientada a objetos e poderosa com diversos recursos incluindo funcionalidades para dispositivos móveis, sendo conhecida como Java Micro Edition (JME). É um robusto e flexível ambiente para desenvolvimento, em conjunto com tecnologias e especificações para criação de plataformas que se adaptem aos requisitos limitados destes dispositivos, tornando possível o desenvolvimento de aplicações Java e fornecendo portabilidade para diferentes tipos de dispositivos móveis. Atualmente o Java ME é responsável pela grande maioria de aplicativos para celulares e PDAs que circulam no mercado. Um aplicativo uma vez escrito pode ser rodado teoricamente em qualquer dispositivo desde que este possua uma máquina virtual instalada. A plataforma Java é

conhecida pelo conceito Write once, run anywhere (Escreva uma vez, rode em qualquer lugar), o que é bastante interessante, pois permite ao programador escrever um código que poderá ser executado em qualquer dispositivo móvel que tenha suporte a Java ME. (Fórum Nokia, 2007).

O trabalho está dividido em 7 capítulos, onde serão abordados conceitos básicos de computação móvel (Capitulo 2), UML (Capitulo 3), Java e JME (Capitulo 4), Sistemas de localização (Capitulo 5), Desenvolvimento do protótipo (Capitulo 6), que descreve todo o processo de desenvolvimento do protótipo como UML e código fonte e o capitulo 7 que descreve as considerações finais sobre os assuntos pesquisados e por fim tem-se as referencias bibliográficas.

1.1. Justificativa

A proposta de desenvolvimento deste projeto é demonstrar a facilidade proporcionada pela tecnologia podendo ser utilizada como informativo sobre condições meteorológicas, tráfego, guia para rotas, obtenção de maior controle de funcionários e veículos de uma empresa, tais como percurso, horários, velocidade, para dispositivos móveis.

1.2. Objetivo geral

O projeto tem como objetivo desenvolver um protótipo de sistema de localização utilizando o GPS (Sistema de Posicionamento Global), direcionado para dispositivos móveis equipados com terminais GPS utilizando a tecnologia Java. O terminal irá receber os dados de satélites GPS, calculando a posição atual do dispositivo e exibindo-os através das coordenadas.

1.3. Objetivos específicos

A seguir são apresentados os passos necessários para a realização do projeto:

- Documentar os conceitos necessários para o desenvolvimento do trabalho: este passo corresponde à fundamentação teórica do trabalho.
- Realizar testes de aplicações com objetos diversos para tela correspondendo aos testes com a plataforma Android.
- Realizar testes com aplicações Android para localização de dados.
- Sincronizar o terminal do dispositivo móvel com os satélites GPS.
- Capturar as informações necessárias para o cálculo da posição do dispositivo.

- Efetuar o cálculo das informações recebidas.
- Exibir o resultado em forma de coordenadas na tela do dispositivo.
- Exibir em um mapa o posicionamento do usuário.
- Localizar no banco de dados os pontos próximos do usuário.
- Exibir uma tela com informações do ponto selecionado.
- Opção de recalculo de melhor escolha de combustível para abastecer.

Documentar a implementação efetuada e fazer os ajustes necessários da fundamentação teórica.

2. COMPUTAÇÃO MÓVEL

A computação móvel está se tornando uma área madura e parece destinada a predominar no futuro. É uma tecnologia que tem como objetivo permitir ao usuário acesso permanente a uma rede fixa ou móvel independente de sua posição física, possibilitando o acesso a informações em qualquer lugar e a qualquer momento, com dispositivos tendo seu próprio disco rígido espaçoso ou que possam se conectar a serviços de armazenamento remoto. (Mateus & Ferreira Loureiro, 2004).

Em plena era da Informação, aplicações de computação móvel podem ser vistas em todas as áreas que requerem mobilidade e comunicação, sem grandes custos e com excelentes vantagens, podendo ser aplicadas em correio eletrônico, comunicação eletrônica via *Paging*, aplicação em automação de vendas, multimídia e aplicações baseadas em GPS.

2.1. Localização de Unidades Móveis

Com o desenvolvimento das redes de comunicação e o aumento da mobilidade surge uma nova necessidade que é acessar serviços dependentes da localização do usuário. Serviços baseados na localização permitem que usuários móveis utilizem serviços baseados na sua posição ou localização geográfica. Este tipo de serviço é usado para definir e prover serviços baseados na posição geográfica de um dispositivo móvel. Normalmente é um serviço em tempo real que pode utilizar o perfil definido pelo assinante para enviar a informação mais adequada naquele momento e local. Deste modo, este serviço pode fornecer informações geo-personalizadas ajudando as pessoas com itinerários e condições de transito, localizando postos de gasolina ou restaurantes, providenciando socorro em caso de emergência, gerenciamento de frotas, rastreamento de cargas e recuperação de veículos roubados, e vários outros serviços.

Um sistema baseado na localização faz uso da localização do usuário, que pode ser física ou semântica. A localização física é definida por algum sistema de coordenadas e a localização semântica específica a localização dentro de um contexto mais amplo, integrando inclusive diferentes fontes de informação.

3. MODELAGEM UML

A UML (*Unified Modeling Language* - Linguagem de Modelagem Unificada) é uma linguagem de modelagem, cujo objetivo é auxiliar os engenheiros de software a definir as características do software, tais como seus requisitos, seus comportamentos, sua estrutura lógica, a dinâmica de seus processos e até mesmo suas necessidades físicas em relação ao equipamento sobre o qual o sistema deverá ser implantado. Todas essas características devem ser definidas por meio da UML antes do software começar a ser desenvolvido.

A modelagem é uma parte central de todas as atividades que levam à implantação de um bom software.

Os modelos fornecem uma copia do projeto de um sistema podendo abranger planos detalhados, assim como planos mais gerais com uma visão panorâmica do sistema considerado. Um bom modelo inclui aqueles componentes que tem ampla repercussão e omite os componentes menores que não são relevantes em determinado nível de abstração. Todos os sistemas podem ser descritos sob diferentes aspectos, com a utilização de modelos distintos, e casa modelo será, portanto, uma abstração semanticamente especifica do sistema. Os modelos podem ser estruturais, dando ênfase à organização do sistema, ou podem ser comportamentais, dando ênfase à dinâmica do sistema.

Com a modelagem são alcançados quatro objetivos:

- Ajudar a visualizar o sistema como ele é ou como desejado
- Permitir especificar a estrutura ou o comportamento de um sistema
- Proporcionar um guia para a construção do sistema
- Documentar as decisões tomadas

4. JAVA

Java é uma tecnologia criada pela *Sun Microsystems* que permite o desenvolvimento de programas para serem usados em qualquer plataforma de processamento que possua uma máquina virtual Java (*Java Virtual Machine*). (Kliemann, 2006).

A primeira edição lançada da plataforma Java foi a *Java Standard Edition* (JSE), que tem como foco o desenvolvimento de aplicações desktop. Posteriormente foi lançada

a edição para servidores *chamada Java Enterprise Edition* (JEE), que visa o desenvolvimento de aplicações servidoras distribuídas e que necessitam de alto desempenho.

Tendo em vista o crescente mercado de aplicações embarcadas, foi criada a edição *Java Micro Edition* (JME), para ser utilizada em dispositivos que abrangem desde TV´s com acesso a Internet até celulares. Atualmente a tecnologia Java conta com mais uma edição chamada *Javacard* que permite a criação de aplicações embarcadas em cartões para diversas finalidades.

A plataforma Java compreende três elementos:

- A linguagem de programação é sintaticamente similar ao C++, mas difere fundamentalmente. Enquanto C++ usa ponteiros inseguros e programadores são responsáveis pela alocação e liberação de memória, a linguagem de programação Java utiliza referências a objetos de tipos seguros, e o não uso da memória é recuperado automaticamente. Além disso, a linguagem de programação Java abstém-se de herança múltipla (uma provável fonte de confusão e ambiguidade no C++) a favor de uma construção límpida, interfaces.
- A máquina virtual forma o fundamento da plataforma Java. Essa arquitetura oferece recursos atrativos: pode ser implementada para executar sobre uma variedade de sistemas operacionais e hardware, com compatibilidade binária aplicações Java operam consistentemente através de muitas implementações. Além disso, a máquina virtual oferece controle firme dos binários executados, oferecendo uma execução segura de código não confiável.
- Finalmente, um conjunto extensível de modelos de interfaces de programação de aplicativos (APIs) rodeia a plataforma Java. Essas APIs suportam inúmeras operabilidades, desde criptografia até localização.

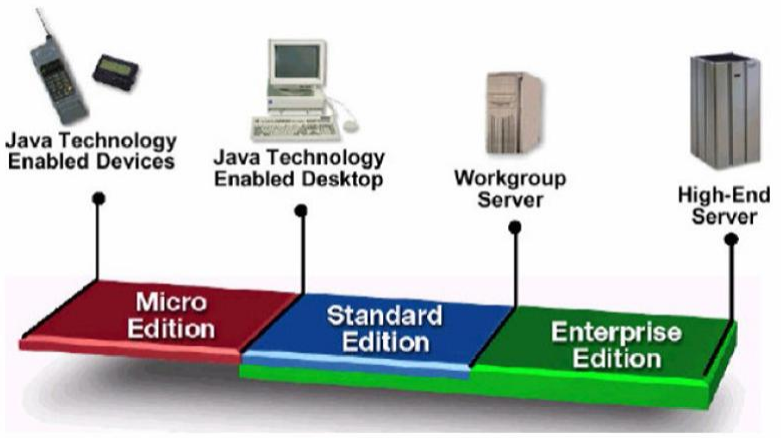

Figura 1 - Elementos da plataforma Java.

4.1. Java Micro Edition

A plataforma *Java Micro Edition* (JME), de forma geral, é a edição de Java para pequenos dispositivos como Pager, telefones celulares, *set-top boxes* de TVs a cabo, PDAs etc. É uma versão específica de máquina virtual criada para ser executada em um

ambiente limitado, com recursos por vezes exíguos de memória e processamento. No cenário atual de mercado, os telefones celulares fazem sombra a outros dispositivos JME devido à sua popularidade, mas é importante lembrar da existência desses outros dispositivos e dos mercados que eles representam. (Kliemann, 2006).

Como ponto importante na tecnologia JME no mercado, destaca-se o fato de que os desenvolvedores são livres para criar aplicações e executá-las em qualquer dispositivo de qualquer fabricante que possua uma máquina virtual, não sendo necessário se prender a um dos fabricantes ou a uma tecnologia. Antes do JME, qualquer programa que precisasse ser incluído em celulares, por exemplo, deveria ser escrito na linguagem nativa do próprio dispositivo.

A plataforma JME é dividida entre *Configurations* (configurações), *Profiles* (perfis) e APIs opcionais. Essa divisão permite ao desenvolvedor conhecer informações específicas sobre as diferentes famílias de dispositivos e as APIs disponíveis em cada uma delas.

4.2. Android

Com o aumento no consumo de dispositivos móveis, empresas e desenvolvedores buscam uma plataforma moderna e ágil para o desenvolvimento de aplicações corporativas para auxiliar em seus negócios e lucros. Já os usuários comuns buscam um dispositivo móvel com um visual elegante e moderno, de fácil navegação e uma infinidade de recursos. Para acompanhar essa evolução da tecnologia e satisfazer os usuários, os fabricantes, operadoras de telefonia celular, as empresas e os desenvolvedores, existe uma grande corrida estrelada pelas maiores empresas do mundo em tecnologia móvel para competir por esse nicho de mercado. (Lecheta, 2009).

Existem muitas opções de sistemas operacionais para smartphones. As principais mais conhecidas são: Blackberry, iPhone OS, Windows Mobile, Symbian e Android. Com essa diversificação de sistemas operacionais existem algumas dificuldades para desenvolvimento de aplicativos para dispositivos móveis, pois cada sistema tem um ambiente de desenvolvimento diferente, onde cada fornecedor de dispositivos móveis tiveram que montar conjuntos de trechos de software de terceiros para criar uma plataforma de telefonia móvel viável. Quase todos os dispositivos que suportam JME também oferecem suporte a extensões proprietárias que limitam a portabilidade das aplicações. Para os desenvolvedores de aplicações para estes dispositivos, significa não ter liberdade para o desenvolvimento, de forma independente, as aplicações móveis criativas que imaginarem, para comercializarem aos proprietários dos dispositivos. (Rogers, Lombardo, Mednieks, & Meike, 2009).

Buscando solucionar a grande maioria deste problemas, algumas empresas, dentre elas a Google, se juntaram e formaram uma aliança para o desenvolvimento de um sistema operacional que soluciona-se estes problemas, chamada de OHA (Open Handset Alliance), lançando assim o Android (OHA, 2009a). Este sistema operacional consiste em uma plataforma de desenvolvimento para aplicativos de dispositivos móveis, com diversas aplicações já instaladas e, ainda, um ambiente de desenvolvimento bastante poderoso, ousado e flexível, onde o gerenciamento da memória, processos, threads, segurança dos arquivos e pastas, além de redes e drivers são gerenciados por sua base no Linux, permitindo que aplicações em segundo plano consigam executar sem que o usuário perceba, enquanto ele está acessando a internet ou atendendo uma ligação. (Lecheta, 2009).

O Android é a primeira plataforma para aplicações móveis completamente livres e de código aberto, o que representa uma grande vantagem para sua evolução, uma vez que diversos programadores do mundo poderão contribuir para melhorar a plataforma. Para os fabricantes de celulares, isso também é uma grande vantagem, uma vez que é possível utilizar o sistema operacional Android em seus dispositivos sem ter que pagar por isso. (Lecheta, 2009).

No Android é utilizada a linguagem Java para construir aplicações, mas o fato interessante é que não existe maquina virtual Java. Na verdade, o que se tem é uma maquina virtual chamada Dalvik que é otimizada para execução em dispositivos móveis. Ao desenvolver aplicações, pode-se utilizar a linguagem Java e todos os seus recursos normalmente, mas depois que o bytecode (.class) é compilado ele é convertido para o formato .dex, que representa a aplicação do Android compilada. Depois disso, os arquivos .dex e outros recursos como imagens são compactados em um único arquivo com a extensão .apk, que representa a aplicação final, pronta para ser distribuída e instalada. (Lecheta, 2009).

A plataforma Android é constituída por vários elementos como aplicações, ferramentas de aplicações, bibliotecas, kernel do Linux, dentre outros, como mostrado na figura 2.

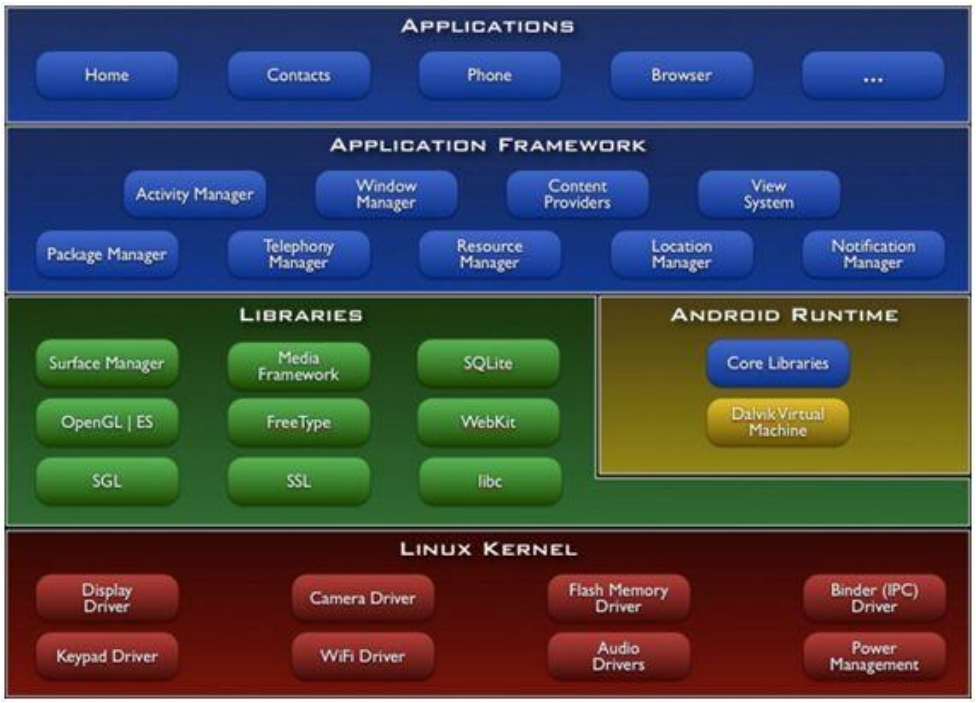

Figura 2 – Elementos da plataforma Android

Na base da pilha está um Linux Kernel versão 2.6. Além de permitir uma abstração entre o hardware e o software, ele é o responsável pelos principais serviços do sistema como gerenciamento de memória e gerenciamento de processos. (Martins, 2009).

A arquitetura foi projetada para simplificar o reuso e a troca de componentes. Para tanto, a camada de framework de aplicação disponibiliza aos desenvolvedores as mesmas APIs (*Application Programming Interface* – Interface de Programação de Aplicativos) usadas para criar as aplicações originais do sistema. (Martins, 2009).

Por padrão, cada aplicação é executada em um processo próprio e cada processo tem a sua máquina virtual. Para cada aplicação é designado um ID de usuário Linux único e as permissões são dadas de forma que os arquivos fiquem visíveis apenas para a aplicação dona. (Martins, 2009).

Uma aplicação não tem um único ponto de entrada. Elas são construídas utilizando componentes que são instanciados no momento em que se tornam necessários. Existem quatro tipos de componentes básicos: (Martins, 2009).

5. SISTEMAS DE LOCALIZAÇÃO

A tecnologia móvel está se tornando extremamente presencial no cotidiano da população envolvendo todas as classes sociais. A presença dos dispositivos móveis está revolucionando a vida das pessoas à medida que esses ficam mais acessíveis e oferecem mais serviços. Nesse sentido, o termo computação móvel é frequentemente empregado no sentido genérico de descrever a habilidade de utilizar tecnologia apropriada para obter

conexão sem fio e usufruir de informação e/ou software de aplicação centralmente encontrado através de aplicações pequenas, portáteis, e dispositivos de computação e comunicação sem fio (Kliemann, 2006).

Serviços baseados em localização (LBS) evoluíram imensamente devido à legislação americana que, em 1996, obrigou as operadoras de telefonia móvel a disponibilizarem a informação de localização de chamadas de emergência realizadas a partir de dispositivos de comunicação (como o otimizado 911 (E911) (ENHANCED, 2006) nos Estados Unidos e a iniciativa do otimizado 112 (E112) na Europa). Esse fato obrigou as operadoras a investirem em infraestrutura que permitisse efetuar esse tipo de localização.

Outros serviços que evoluíram significativamente nos últimos anos melhorando as condições dos serviços baseados em localização foram o GPS, criado e sustentado pelo sistema de defesa militar do governo dos Estados Unidos, e os GIS, sistemas de informação geográfica disponibilizados por diversas empresas.

Acompanhando a evolução dos serviços baseados em localização e da crescente oportunidade de capitalizar no mercado, empresas de Tecnologia da Informação, como a *Sun Microsystems* que criou a plataforma J2ME e a linguagem de programação Java, procuraram disponibilizar interfaces de programação de aplicativos (APIs) que capacitassem programadores a desenvolver aplicações para dispositivos móveis capazes de obter, manipular, armazenar e distribuir a sua localização. A API disponibilizada pela *Sun* denomina-se "*Location*" ou "JSR-179".

5.1. GPS

É um aparelho de radio especial que mede a localização a partir de satélites que orbitam a Terra e que transmitem essa informação por sinais de radio. (Vitorelli, 2009).

Consiste de três órbitas medias com oito satélites cada. As órbitas foram projetadas de forma que cada ponto da Terra seja capaz de enxergar pelo menos quatro satélites em qualquer instante do dia.

Para descobrir a sua posição, o receptor GPS sabe que cada satélite se encontra na superfície de uma esfera imaginaria centrado no próprio satélite e com raio igual a distancia calculada. A intersecção de duas esferas forma um círculo, estando então o receptor em um ponto dentro deste circulo. Se uma terceira esfera for usada, ela cortará o circulo em dois pontos, logo o receptor se encontra em algum desses dois pontos. Uma quarta esfera determina qual desses pontos é a posição correta em relação aos quatros satélites.

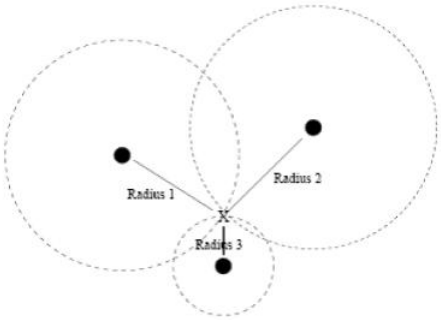

Figura 3 – Método de triangulação

Uma vez obtido os dados do GPS, o mesmo deve formatá-los no padrão NMEA-0183 (*National marines eletronic association*). Atualmente existem vinte e seis tipos de mensagens neste formato. A lógica nas mensagens segue o padrão, cada sentença começa com o símbolo $ e termina com *CARRIGE RETURN* e *LINE FEED*. Os dados são separados por vírgula, portanto os números não usam ponto e vírgula para separar as casas decimais e sim ponto.

As principais mensagens neste formato que são apresentadas em um dispositivo GPS são:

- GPGGA (global positioning system fix data)
- GPGSA (GPS DOP and active satellites)
- GPGSV (GPS satellites in view)
- GPRMC (recommended minimum specific GPS/TRANSIT data)

Uma mensagem no padrão NMEA-0183 em formato GPRMC pode ser exemplificada conforme a figura abaixo:

$GPRMC,081836,A,3751.65,S,14507.36,E,000.0,360.0,130998,011.3,W,A*62
1 2 3 4 5 6 7 8 9 10 11 12 13

Leganda:
1 - Hora
2 - Status da Mensagem (A=dados válidos, V=dados inválidos)
3 - Latitude
4 - N = Norte ou S = Sul da longitude
5 - Longitude
6 - E = Leste ou W = Oeste
7 - Velocidade em nós
8 - Direção do movimento em relação ao norte (000.0 a 359.9 graus)
9 - Data
10 - Variação Magnética
11 - E = Leste ou W = Oeste da Variação Magnética
12 - Indicador, (A=Autônomo, D=Diferente, E=Estimado, N=Dados inválidos)
13 - Código verificador

Figura 4 - Mensagem no padrão NMEA-0183 em formato GPRMC

5.2. Características

Métodos baseados nos dispositivos

- *Global Positioning System* (GPS): Desenvolvido pelo sistema de defesa dos Estados Unidos e mantido por uma facção da força aérea desse país. Esse método é baseado no sinal transmitido por 24 satélites em órbita com a Terra (e mais três em caso de falhas) a aproximadamente 20.000 km da superfície. São necessários três satélites para obtenção de localização bidimensional (2D) como latitude e longitude e quatro satélites para tridimensional (3D) como altitude. A sua precisão varia entre um cm e 30 metros, dependendo do aparelho utilizado, do sistema utilizado (público, da velha ou nova geração; ou militar), das condições do clima e do local onde se está adquirindo o sinal. Apresenta sinal fraco em prédios e grandes centros metropolitanos onde há lugares cercados por concreto e prédios altos. As suas principais características: cobertura global (incluindo regiões polares), disponibilidade contínua, serviço passivo (apenas recebe a informação), suporta número ilimitado de usuários, alta precisão e localização expressa em latitude e longitude. (Kliemann, 2006).
- *Enhanced Observed Time Difference* (E-OTD): Similar ao TDOA, mas a posição é estimada pelo aparelho, não pela estação base. Exige software dedicado no terminal e nas bases. Precisão relativamente alta, entre 50 e 200 metros (GSM, 2006), mas aproxima-se do GPS somente em áreas urbanas, onde diversas células estão disponíveis. • *Subscriber Identity Module Toolkit* (SIM): O SIM Toolkit é uma API que permite a comunicação com o *smartcard* SIM, o qual muitos telefones celulares possuem de aplicações que foram instaladas no dispositivo. A qualidade pode ser tão ruim quanto o método COO, mas pode também ser incrementado por alguns algoritmos que são armazenados no SIM e recursos da operadora.
- Bluetooth;
- Infrared (Infravermelho).

Métodos baseados na rede

- Cell of Origin (COO): É o método mais comum e mais fácil, mas também o mais impreciso. Precisão de 200 metros em áreas urbanas, 2 km em suburbanas e 3 a 4 km em zonas rurais (GSM, 2006). A rede determina apenas a célula na qual o usuário se encontra. (Kliemann, 2006).
- *Angle of Arrival* (AOA): Esse método utiliza equipamento especial que precisa ser instalado nas estações base para determinar o ângulo de chegada do sinal de rádio. Com alguns cálculos básicos pode-se localizar com apenas duas antenas. *Time Difference of Arrival* (TDOA) ou *Time of Arrival* (TOA): Baseado na diferença de chegada dos sinais de rádio na estação base do dispositivo. Exige no mínimo três estações base.
- *Location Pattern of Matching* (LPM): Esse método complexo analisa o sinal de rádio e compara com modelos armazenados em base de dados. Esses modelos incluem sinais de reflexão e eco. Quando um modelo é reconhecido, a localização é obtida. Esse método só pode ser utilizado em áreas urbanas, onde os sinais são frequentes. Nessas áreas a qualidade deve ser melhor que outros métodos, mas infelizmente não pode ser

aplicado em áreas rurais.

Método hibrido

- *Assisted GPS* (A-GPS): Utiliza informações da rede para determinar mais rapidamente a posição dos quatro satélites que a serem ouvidos. A célula da rede distribui a localização desses satélites e pode drasticamente reduzir o tempo de inicialização que receptores GPS precisam. Além disso, esse método economiza bateria já que o receptor GPS somente é acionado em áreas úteis. (Kliemann, 2006).

6. DESENVOLVIMENTO DO PROTÓTIPO

Como estudo de caso será desenvolvido um aplicativo baseado em localização chamado de Gasosa, onde o usuário poderá localizar as coordenadas de sua posição atual e verificar os pontos cadastrados próximo a sua localização e o tipo de combustível mais viável para abastecer. Abaixo serão exibidas as UMLs e a codificação do protótipo.

6.1. UML

Para um melhor entendimento do desenvolvimento do protótipo proposto, abaixo seguem os diagramas de caso de uso, de classe e de sequência.

Diagrama de caso de uso

A figura 5 exibe o diagrama de caso de uso do desenvolvimento do protótipo proposto.

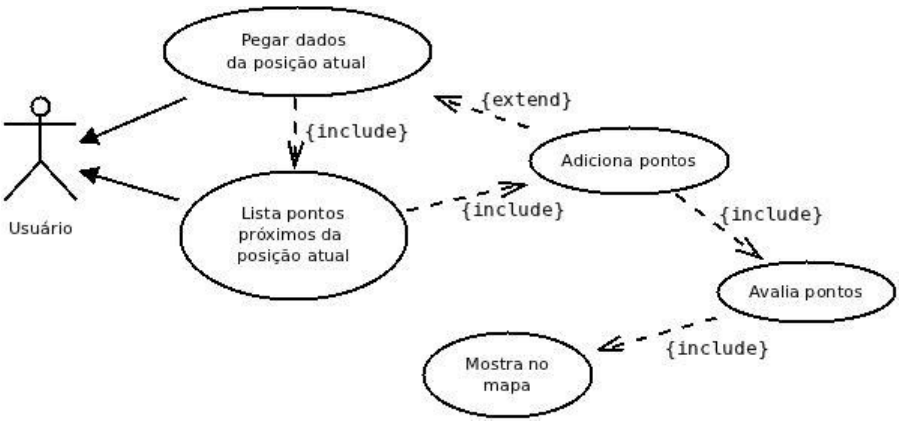

Figura 5 - Diagrama de Caso de Uso do Protótipo

Diagrama de classe

A figura 6 ilustra o diagrama de classe do protótipo Gasosa com suas respectivas classes.

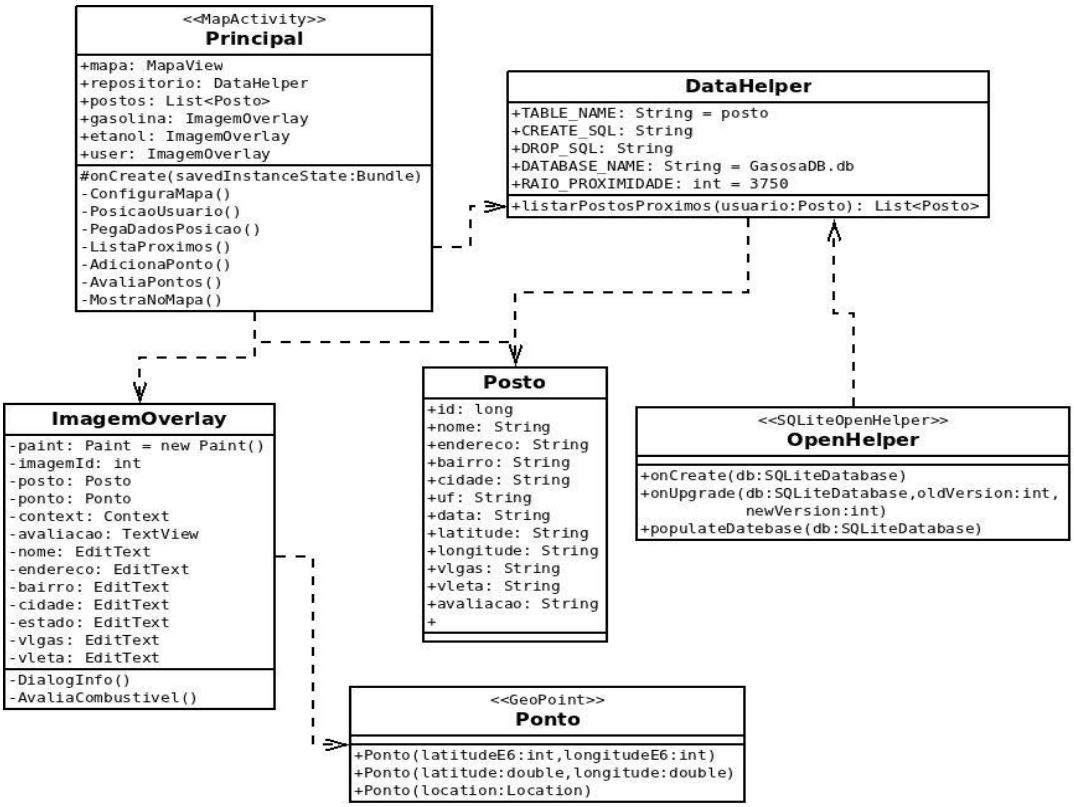

Figura 6 - Diagrama de Classe do protótipo

Fluxograma

Conforme mostrado na figura 7 abaixo, vemos o fluxograma do protótipo Gasosa.

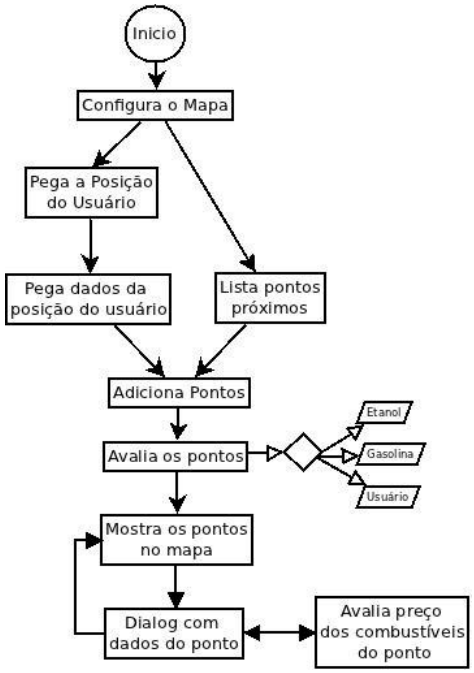

Figura 7 - Diagrama de sequencia do protótipo

6.2. CODIFICAÇÃO DO PROTOTIPO

O protótipo desenvolvido tem com o nome de Gasosa. Este protótipo de sistema possui algumas classes, tais como:

Classe Principal:

Esta é a classe principal do protótipo, responsável pela configuração do mapa, pegar a posição do usuário e dados de sua localização, listar os pontos em sua proximidade e separar os tipos de pontos. Esta classe possui alguns métodos, conforme descrito abaixo:

- **Método *ConfiguraMapa***: Método responsável pela configuração do mapa, como nível do zoom e permitir o clique no mapa.

```
72    private void ConfiguraMapa() {
73        mapa = (MapView) findViewById(R.id.mapa);
74        mapa.setClickable(true);
75        controlador = mapa.getController();
76
77        controlador.setZoom(20);
78        ZoomControls zoom = (ZoomControls) mapa.getZoomControls();
79        zoom.setLayoutParams(new ViewGroup.LayoutParams(ViewGroup.LayoutParams.FILL_PARENT, ViewGroup.LayoutParams.FILL_PARENT));
80        zoom.setGravity(Gravity.BOTTOM + Gravity.CENTER_HORIZONTAL);
81        mapa.addView(zoom);
82        mapa.displayZoomControls(true);
83
84    }
```

Listagem 1 - Método *ConfiguraMapa*

- **Método *PosicaoUsuario:*** Método responsável por pegar a posição do usuário e centralizar o mapa em sua posição atual.

```
86    private void PosicaoUsuario() {
87        loc = getLocationManager().getLastKnownLocation(LocationManager.GPS_PROVIDER);
88
89        if (loc != null) {
90            ponto = new Ponto(loc);
91            Log.i(LOG_TAG, "Localização: " + loc.getLatitude() + ", " + loc.getLongitude());
92            controlador.setCenter(ponto);
93        }
94
95
96        // usuario.onTap(ponto, mapa);
97
98        getLocationManager().requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0, this);
99
100   }
```

Listagem 2 - Método PosicaoUsuario

- **Método *PegaDadosPosicao*:** Este método é responsável por pagar os dados como endereço, bairro, cidade e estado referente as coordenadas da posição atual do usuário.

```
102    private void PegaDadosPosicao() {
103
104        Geocoder gc = new Geocoder(this, Locale.getDefault());
105
106        try {
107            List<Address> addresses = gc.getFromLocation(loc.getLatitude(), loc.getLongitude(), 1);
108
109            if (addresses.size() > 0) {
110                Address address = addresses.get(0);
111
112                // endereco = address.getAddressLine(0);
113
114                novoposto.endereco = address.getAddressLine(0).substring(0, address.getAddressLine(0).lastIndexOf("-"));
115                novoposto.bairro = address.getAddressLine(0).substring(address.getAddressLine(0).lastIndexOf("-") + 1);
116                novoposto.cidade = address.getLocality().toString();
117                novoposto.uf = address.getAdminArea().toString();
118                novoposto.latitude = String.valueOf(loc.getLatitude());
119                novoposto.longitude = String.valueOf(loc.getLongitude());
120                novoposto.avaliacao = "U";
121
122                Log.i(LOG_TAG, "Endereço: " + novoposto.endereco);
123            }
124        } catch (Exception e) {
125            Log.i(LOG_TAG,
126                    "Não foi possível encontrar dados da localização do usuário." + e);
127        }
128    }
```

Listagem 3 - Método PegaDadosPosicao

- **Método *AvaliaPontos*:** Este método separa quais pontos estão avaliados no vetor postos como melhor combustível, se gasolina ou etanol ou usuário, colocando uma imagem diferente para cada tipo de avaliação.

```
144    private void AvaliaPontos(){
145
146        for (int i = 0; i < postos.size(); i++){
147 //         Log.i(LOG_TAG, "AVALIACAO: " + i + " - " + postos.get(i).avaliacao);
148            ponto = new Ponto(Double.parseDouble(postos.get(i).latitude), Double.parseDouble(postos.get(i).longitude));
149
150            if (postos.get(i).avaliacao.equals("G")){
151                gasolina = new ImagemOverlay(postos.get(i), R.drawable.gasolina, this);
152                mapa.getOverlays().add(gasolina);
153                mapa.getController().setCenter(ponto);
154 //             MostraNoMapa(gasolina);
155
156            } else if(postos.get(i).avaliacao.equals("E")){
157                etanol = new ImagemOverlay(postos.get(i), R.drawable.etanol, this);
158                mapa.getOverlays().add(etanol);
159 //             MostraNoMapa(etanol);
160
161            } else if(postos.get(i).avaliacao.equals("U")){
162                user = new ImagemOverlay(postos.get(i), R.drawable.usuario, this);
163                mapa.getOverlays().add(user);
164 //             MostraNoMapa(user);
165 //             usuario = new MyLocationOverlay(this, mapa);
166 //             mapa.getOverlays().add(usuario);
167 //             mapa.getController().setCenter(ponto);
168
169            } else {
170                semavaliacao = new ImagemOverlay(postos.get(i), R.drawable.semavaliacao, this);
171                mapa.getOverlays().add(semavaliacao);
172
173 //             MostraNoMapa(semavaliacao);
174            }
175
176        }
177    }
```

Listagem 4 - Método AvaliaPontos

Classe DataHelper:

Classe responsável pela manipulação do banco de dados. Nela é feita a listagem dos pontos próximos a posição do usuário.

- **Método de *listarPostosProximos*:** Método responsável pela filtragem dos

pontos próximos da posição do usuário, registrados no banco de dados. Na listagem abaixo podemos verificar que o método recebe as coordenadas da posição do usuário e realiza a filtragem comparando se distancia do registro atual está dentro do raio de proximidade pré-definido pelo sistema, caso esteja, o registro é inserido em um vetor do mesmo tipo para retorno a classe Principal.

```
114  public List<Posto> listarPostosProximos(Posto usuario) {
115      Cursor c = getCursor();
116      List<Posto> postos = new ArrayList<Posto>();
117      if (c.moveToFirst()) {
118          do {
119              Posto posto = new Posto();
120              posto.id = c.getLong(c.getColumnIndex(Postos._ID));
121              posto.nome = c.getString(c.getColumnIndex(Postos.NOME));
122              posto.endereco = c.getString(c.getColumnIndex(Postos.ENDERECO));
123              posto.bairro = c.getString(c.getColumnIndex(Postos.BAIRRO));
124              posto.cidade = c.getString(c.getColumnIndex(Postos.CIDADE));
125              posto.uf = c.getString(c.getColumnIndex(Postos.UF));
126              posto.data = c.getString(c.getColumnIndex(Postos.DATA));
127              posto.latitude = c.getString(c.getColumnIndex(Postos.LATITUDE));
128              posto.longitude = c.getString(c.getColumnIndex(Postos.LONGITUDE));
129              posto.vlgas = c.getString(c.getColumnIndex(Postos.VLGASOLINA));
130              posto.vleta = c.getString(c.getColumnIndex(Postos.VLETANOL));
131              posto.vleta = c.getString(c.getColumnIndex(Postos.VLETANOL));
132              posto.avaliacao = c.getString(c.getColumnIndex(Postos.AVALIACAO));
133
134              double latP = Double.parseDouble(posto.latitude.toString());
135              double lonP = Double.parseDouble(posto.longitude.toString());
136              Location locPosto = new Location("reverseGeocoded");
137              locPosto.setLatitude(latP);
138              locPosto.setLongitude(lonP);
139
140              double latU = Double.parseDouble(usuario.latitude);
141              double lonU = Double.parseDouble(usuario.longitude);
142              Location locUsuario = new Location("reverseGeocoded");
143              locUsuario.setLatitude(latU);
144              locUsuario.setLongitude(lonU);
145
146              int distancia = (int) locUsuario.distanceTo(locPosto);
147
148              if (distancia <= RAIO_PROXIMIDADE) {
149                  postos.add(posto);
150              }
151          } while (c.moveToNext());
152      }
153      return postos;
154  }
```

Listagem 5 - Método listarPostosProximos

Classe ImagemOverlay:

Classe responsável pela sobreposição dos pontos sobre o mapa, exibição da tela com as informações do ponto clicado e recalculo de qual o melhor combustível para se abastecer.

- **Método *draw*:** Método responsável por pintar no mapa a imagem setada na posição de suas coordenadas.

```
55   public void draw(Canvas canvas, MapView mapView, boolean shadow) {
56       super.draw(canvas, mapView, shadow);
57
58       Point p = mapView.getProjection().toPixels(ponto, null);
59       Bitmap btm = BitmapFactory.decodeResource(mapView.getResources(), this.imagemId);
60       RectF r = new RectF(p.x, p.y, p.x + btm.getWidth(), p.y
61               + btm.getHeight());
62       canvas.drawBitmap(btm, null, r, paint);
63   }
```

Listagem 6 - Método draw

- **Método *onTap*:** Este método é o responsável por verificar se o ponto clicado no mapa corresponde a algum dos pontos listados. Caso afirmativo, este método chama o método DialogInfo.

```
71  public boolean onTap(GeoPoint geoPoint, MapView mapView) {
72      Point ponto = mapView.getProjection().toPixels(this.ponto, null);
73      // Cria o ret?ngulo
74      RectF recf = new RectF(ponto.x - 5, ponto.y - 5, ponto.x + 5, ponto.y + 5);
75      // Converte para ponto em pixels
76      Point newPoint = mapView.getProjection().toPixels(geoPoint, null);
77      // Verifica se o ponto est? contido no ret?ngulo
78      boolean ok = recf.contains(newPoint.x, newPoint.y);
79      if (ok) {
80          // Toast.makeText(mapView.getContext(), "Nome do Posto: " +
81          // posto.nome + " - Endereço: " + posto.endereco,
82          // Toast.LENGTH_SHORT).show();
83          // return true;
84          DialogInfo();
85          Log.i(LOG_TAG, "AVALIACAO: " + posto.avaliacao);
86      }
87      return super.onTap(geoPoint, mapView);
88  }
```

Listagem 7 - Método onTap

- **Método *DialogInfo*:** Este método exibe por cima do mapa uma tela com os dados do ponto clicado, além de disponibilizar a opção de recalcular qual o combustível mais viável de se abastecer.

```
90      private void DialogInfo() {
91          final Dialog dialog = new Dialog(context);
92          dialog.setContentView(R.layout.dialoginfo);
93          dialog.setTitle("Informaçoes do Ponto");
94          dialog.setCancelable(true);
95
96          nome = (EditText) dialog.findViewById(R.id.nome);
97          endereco = (EditText) dialog.findViewById(R.id.endereco);
98          bairro = (EditText) dialog.findViewById(R.id.bairro);
99          cidade = (EditText) dialog.findViewById(R.id.cidade);
100         estado = (EditText) dialog.findViewById(R.id.estado);
101         vlgas = (EditText) dialog.findViewById(R.id.vlgas);
102         vleta = (EditText) dialog.findViewById(R.id.vleta);
103         avaliacao = (TextView) dialog.findViewById(R.id.avaliacao);
104
105         nome.setText(posto.nome);
106         endereco.setText(posto.endereco);
107         vlgas.setText(posto.vlgas);
108         vleta.setText(posto.vleta);
109         estado.setText(posto.uf);
110         cidade.setText(posto.cidade);
111         bairro.setText(posto.bairro);
112
113         Button calcular = (Button) dialog.findViewById(R.id.btnCalcular);
114
115         calcular.setOnClickListener(new View.OnClickListener() {
116             @Override
117             public void onClick(View arg0) {
118                 AvaliaCombustivel();
119             }
120         });
121
122         dialog.show();
123     }
```

Listagem 8 - Método DialogInfo

- **Método *AvaliaCombustivel*:** Este método pega os dados dos campos de gasolina e etanol na tela chamada pelo método DialogInfo e realiza o calculo de viabilidade dos combustíveis e informando na tela qual o melhor para se abastecer.

```
125    private void AvaliaCombustivel() {
126        Double gasolina = 0.0;
127        Double etanol = 0.0;
128
129        gasolina = Double.parseDouble(vlgas.getText().toString());
130        etanol = Double.parseDouble(vleta.getText().toString());
131
132        if (gasolina / 100 * 70 <= etanol) {
133            posto.avaliacao = "G";
134            avaliacao.setText("Melhor abastecer com GASOLINA");
135        } else {
136            posto.avaliacao = "E";
137            avaliacao.setText("Melhor abastecer com ETANOL");
138        }
139    }
```

Listagem 9 - Método AvaliaCombustivel

Classe OpenHelper:

Classe responsável pela sobreposição dos pontos sobre o mapa, exibição da tela com as informações do ponto clicado e recalculo de qual o melhor combustível para se abastecer.

- **Método *onCreate*:** Método responsável por criar o banco de dados do protótipo.

```
16     public void onCreate(SQLiteDatabase db) {
17         Log.i(DataHelper.LOG_TAG, "Creating database");
18         db.execSQL(DataHelper.CREATE_SQL);
19         db.setVersion(DataHelper.DATABASE_VERSION);
20         populateDatabase(db);
21     }
```

Listagem 10 - Método onCreate

- **Método *onUpgrade*:** Método responsável por atualizar o banco de dados do prototipo.

```
24  public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
25
26      Log.w(DataHelper.LOG_TAG, "Upgrading database, this will drop tables and recreate.");
27      db.execSQL(DataHelper.DROP_SQL);
28
29      Log.i(DataHelper.LOG_TAG, "Creating database");
30      db.execSQL(DataHelper.CREATE_SQL);
31      db.setVersion(newVersion);
32      populateDatabase(db);
33  }
```

Listagem 11 - Método onUpgrade

- **Método *populateDatabase*:** Método responsável por preencher o banco de dados com dados pré-estipulados.

```
35    private void populateDatabase(SQLiteDatabase db) {
36        db.beginTransaction();
37
38        db.execSQL("INSERT INTO posto (nome, endereco, bairro, cidade, uf, data, latitude, longitude, vlgas, vleta, avaliacao) VALUES " +
39            "('Posto 1' ,'Rua 20','Centro'          ,'Campo Grande','MS','15/06/2011', '-20.4579', '-54.6090', '2.333','1.989', 'G')");
40        db.execSQL("INSERT INTO posto (nome, endereco, bairro, cidade, uf, data, latitude, longitude, vlgas, vleta, avaliacao) VALUES " +
41            "('Posto 2' ,'Rua 19','Carandá Bosque I'    ,'Campo Grande','MS','15/06/2011', '-20.4578', '-54.6091', '2.334','1.988', 'E')");
42        db.execSQL("INSERT INTO posto (nome, endereco, bairro, cidade, uf, data, latitude, longitude, vlgas, vleta, avaliacao) VALUES " +
43            "('Posto 3' ,'Rua 18','Jardim dos Estados','Campo Grande','MS','15/06/2011', '-20.4577', '-54.6092', '2.335','1.987', 'G')");
44        db.execSQL("INSERT INTO posto (nome, endereco, bairro, cidade, uf, data, latitude, longitude, vlgas, vleta, avaliacao) VALUES " +
45            "('Posto 4' ,'Rua 17','Monte Castelo'       ,'Campo Grande','MS','15/06/2011', '-20.4576', '-54.6093', '2.336','1.986', 'E')");
46        db.execSQL("INSERT INTO posto (nome, endereco, bairro, cidade, uf, data, latitude, longitude, vlgas, vleta, avaliacao) VALUES " +
47            "('Posto 5' ,'Rua 16','Centro'          ,'Campo Grande','MS','15/06/2011', '-20.4575', '-54.6094', '2.337','1.985', 'G')");
48        db.execSQL("INSERT INTO posto (nome, endereco, bairro, cidade, uf, data, latitude, longitude, vlgas, vleta, avaliacao) VALUES " +
49            "('Posto 6' ,'Rua 15','Jardim Petropolis' ,'Campo Grande','MS','15/06/2011', '-20.4574', '-54.6095', '2.338','1.984', 'E')");
50        db.execSQL("INSERT INTO posto (nome, endereco, bairro, cidade, uf, data, latitude, longitude, vlgas, vleta, avaliacao) VALUES " +
51            "('Posto 7' ,'Rua 14','Centro'          ,'Campo Grande','MS','15/06/2011', '-20.4573', '-54.6096', '2.339','1.983', 'G')");
52        db.execSQL("INSERT INTO posto (nome, endereco, bairro, cidade, uf, data, latitude, longitude, vlgas, vleta, avaliacao) VALUES " +
53            "('Posto 8' ,'Rua 13','Carandá'         ,'Campo Grande','MS','15/06/2011', '-20.4572', '-54.6097', '2.340','1.982', 'E')");
54        db.execSQL("INSERT INTO posto (nome, endereco, bairro, cidade, uf, data, latitude, longitude, vlgas, vleta, avaliacao) VALUES " +
55            "('Posto 9' ,'Rua 12','Centro'          ,'Campo Grande','MS','15/06/2011', '-20.4571', '-54.6098', '2.341','1.981', 'G')");
56        db.execSQL("INSERT INTO posto (nome, endereco, bairro, cidade, uf, data, latitude, longitude, vlgas, vleta, avaliacao) VALUES " +
57            "('Posto 10','Rua 11','Centro'          ,'Campo Grande','MS','15/06/2011', '-20.4570', '-54.6099', '2.342','1.980', 'E')");
58
59        db.setTransactionSuccessful();
60        db.endTransaction();
61    }
```

Listagem 12 - Método populateDatabase

Classe Posto:

Classe do tipo entidade, responsável pela transição de dados entre classes.

- **Método *Posto*:** Método responsável por transmitir os dados do ponto.

```
35    protected Posto(long id, String nome, String endereco, String bairro,
36            String cidade, String uf, String data, String latitude,
37            String longitude, String vlGas, String vlEta, String avaliacao) {
38        super();
39        this.id = id;
40        this.nome = nome;
41        this.endereco = endereco;
42        this.bairro = bairro;
43        this.cidade = cidade;
44        this.uf = uf;
45        this.data = data;
46        this.latitude = latitude;
47        this.longitude = longitude;
48        this.vlgas = vlGas;
49        this.vleta = vlEta;
50        this.avaliacao = avaliacao;
51
52    }
```

Listagem 13 - Método Posto

Classe Ponto:

Classe responsável pela conversão das coordenadas.

- **Método *Ponto*:** Método responsável por converter das coordenadas recebidas de double para int.

```
 9    public Ponto(int latitudeE6, int longitudeE6) {
10        super(latitudeE6, longitudeE6);
11    }
```

Listagem 14 - Método Ponto

Pode-se executar também o emulador do Android, que é utilizado para efetuar os testes do protótipo. A tela de execução neste emulador é a mostrada a seguir

- **Tela *Principal*:** Nesta tela é apresentada ao usuário a sua localização atual e os postos próximos dentro do raio de sua posição.

Figura 8 - Tela Principal

- **Tela *DialogInfo*:** Ao clicar em cima de qualquer ponto marcado no mapa, abrirá uma tela com as informações do ponto clicado, possibilitando um novo cálculo de qual a melhor escolha para abastecimento.

Figura 9 - Tela DialogInfo com dados do ponto clicado

- **Tela *Resultado*:** Na mesma tela DialogInfo, quando digitado um novo valor no campos gasolina e etanol e pressionado o botão calcular, logo abaixo será exibido uma mensagem com a mesmo escolha para se

abastecer, conforme mostrado na figura abaixo.

Figura 10 - Tela DialogInfo exibindo o resultado de viabilidade

7. CONSIDERAÇÕES FINAIS

As ferramentas de localização utilizadas por empresas cada vez mais precisam atender a serviços e processos de escala global, com velocidade, rapidez e, muitas vezes, em tempo real. Diversos segmentos como transportadoras, segurança patrimonial, engenharia de trafego, dentre outros são usuários de ferramentas de localização para monitoração de frotas, vigilâncias, localização e mapeamento de rotas, entre outros serviços. Para a execução de tais processos, muitas vezes, é necessária a utilização de diversos equipamentos e sistemas que podem, em muitos casos, não serem compatíveis, possuírem tecnologia ultrapassada ou necessitar ainda de interação humana, o que potencializa possíveis falhas no sistema. (Vitorelli, 2009).

Seja através de satélites GPS ou de sistemas terrestres, serviço baseados em localização ou LBS podem auxiliar os automobilistas, orientar os turistas, ou informar o publico em geral sobre características e serviços referentes à determinada zona ou região. Além disso, é possível agregar valor aos serviços de localização, por meio de informações e serviços de orientação que auxiliem na localização de um determinado dispositivo móvel.

Existe um segmento em crescimento chamado geobusiness, que envolve planejamento de logística para empresas interessadas em serviços focado em planejamento de rotas, mapeamento e localização de cargas especificas do setor. Além disso, favorecendo a integração e crescimento deste setor, diversas empresas fabricantes de celulares estão iniciando operações comerciais de navegação no Brasil oferecendo

celulares com recursos PGS integrados. Juntamente com a plataforma JME nos celulares, com suporte à API JSR 179 de localização e posicionamento, é possível realizar a integração entre aplicações e GPS.

A conclusão desta pesquisa foi a contribuição na área de sistemas distribuídos, em especial para dispositivos moveis tais como: PDAs, celulares, smartphones, dentre outros, com a aplicação de mecanismos de posicionamento global em plataforma Android.

Como trabalhos futuros, poderá ser realizada a implementação de sincronização com um servidor remoto para troca de dados entre registros do banco do dispositivo móvel e o banco de dados do servidor online, podendo ser compartilhado assim com outros usuários do aplicativo, além de cadastro de novos pontos, alterações de dados dos registros e remoções de pontos.

REFERÊNCIAS (ESTILO <SECAOSEMNUM>)

Booch, G., Rumbaugh, J., & Jacobson, I. (2000). UML - Guia do Usuário. Rio de Janeiro: Campus.

Brito, R. C. (Diretor). (2009). Conectando aplicações JME com servidores remoto [Filme Cinematográfico].

Dilão, R. (s.d.). Sistema de Posicionamento Global (GPS). Acesso em 03 de 2009, disponível em Longitude e Latitude instrumentos e medição: http://www.cienciaviva.pt/latlong/anterior/gps.asp

Ferrari, B. (21 de Janeiro de 2009). 6 tendências tecnológicas para 2009. Acesso em 03 de 2009, disponível em Info Professional: http://info.abril.com.br/professional/tendencias/6-tendencias-tecnologicas-para.shtml?3

Forum Nokia. (24 de Outubro de 2007). Java ME. Acesso em 03 de 2009, disponível em Forum Nokia: http://wiki.forum.nokia.com/index.php/Java_ME_-_Portugu%C3%AAs

Guedes, G. T. (2006). UML - Uma Abordagem Prática. São Paulo: Novatec.

Java ME at a Glance. (s.d.). Acesso em 03 de 2009, disponível em Sun Developer Network (SDN): http://java.sun.com/javame/index.jsp

Kliemann, J. M. (Junho de 2006). Modelagem de uma API para serviços baseados em localização integrada com APIs de localização, o middleware EXEHDA e GIS. Porto Alegre.

Lecheta, R. R. (2009). Google Android: aprenda a criar aplicações para dispositivos móveis com o Android SDK. São Paulo: Novatec.

Lima, L. A. (03 de 07 de 2000). Acesso em 16 de 04 de 2009, disponível em COMPUTAÇÃO MÓVEL: http://www.dimap.ufrn.br/~gold/CMovel.html

Loureiro, A. A. (s.d.). Comunicação sem fio e Computação Móvel: Tecnologias, desafios e oportunidades.

Loureiro, A. A., Sadok, D. F., Mateus, G. R., Nogueira, J. M., & Kelner, J. (Agosto de 2003). Comunicação sem fio e Computação Móvel: Tecnologias, Desafios e Oportunidades. Campinas, São Paulo.

Martins, R. J. Desenvolvimento de Aplicativo para Smartphone com a Plataforma Android. PROJETO FINAL DE GRADUAÇÃO. PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO, Rio de Janeiro.

Mateus, G. R., & Ferreira Loureiro, A. A. (2004). Introdução à Computação Móvel. Rio de Janeiro.

NOKIA Corporation. (s.d.). Manual do Usuário. NOKIA N95-2.

Ogliari, R. (Maio de 2009). Criando um aplicativo LBS para consulta de pontos turísticos em cidades. Web Services + Mobilidade, pp. 22 - 27.

Penna, G. (27 de Feveveiro de 2009). Carreira em mobilidade está em alta. Acesso em 03 de 2009, disponível em Info Professional: http://info.abril.com.br/professional/carreira/mobilidade-em-alta.shtml

Reis, C., Carmo, M., & Soto, C. (s.d.). Computação Móvel:Tópicos Essenciais. Acesso em 16 de 04 de 2009, disponível em Computação Móvel:Tópicos Essenciais: http://www.async.com.br/~kiko/mobilcomp/0.php

Rogers, R., Lombardo, J., Mednieks, Z., & Meike, B. (2009). Desenvolvimento de aplicações Android: programação com o SDK do Google. São Paulo: O'Reilly - Novatec.

SIlva, M. (Novembro de 2007). Desenvolvimento de aplicações para dispositivos moveis.

Strickland, J. (s.d.). O futuro da computação móvel. Acesso em 16 de 04 de 2009, disponível em Uol Informática: http://informatica.hsw.uol.com.br/netbooks-notebooks-umpcs3.htm

Sun Microsystems. (s.d.). Tecnologia Java ME. Acesso em 04 de 04 de 2009, disponível em Sun Microsystems: http://java.sun.com/javame/technology/index.jsp

Valente, S. D. (2005). TECNOLOGIA J2ME: JAVA 2 MICRO EDITION. Minas Gerais.

Vitorelli, A. (09 de 03 de 2009). Framework Java em ambiente distribuido com posicionamento global. São Paulo.

Ricardo Lúcio Resende Freitas Junior

Nascido no Rio de Janeiro em 1977. Mudou-se para Campo Grande/MS em 2004. Iniciou sua carreira em uma empresa onde atuava como estagiário de informática em 2000, responsável em desenvolver o portal da empresa, e nos anos seguintes em outras empresas como desenvolvedor e responsável técnico pela manutenção dos computadores, adquirindo considerável conhecimento tanto em programação como em manutenção de microcomputadores.

coautor1

biografia...