UNIVERSIDADE PARA O DESENVOLVIMENTO DA REGIÃO E DO ESTADO DO PANTANAL – UNIDERP

CURSO DE ENGENHARIA DA COMPUTAÇÃO

RICARDO LÚCIO RESENDE FREITAS JUNIOR Nº RA 51056

LOCALIZAÇÃO VIA SATELITE DESENVOLVIDO PARA DISPOSITIVOS MÓVEIS UTILIZANDO A PLATAFORMA ANDROID

CAMPO GRANDE – MS

2011

RICARDO LÚCIO RESENDE FREITAS JUNIOR

51056

LOCALIZAÇÃO VIA SATELITE DESENVOLVIDO PARA DISPOSITIVOS MÓVEIS UTILIZANDO A PLATAFORMA ANDROID

Monografia apresentada como exigência para a disciplina de Trabalho Final de Graduação II do curso de Engenharia da Computação da Universidade para o Desenvolvimento do Estado e da Região do Pantanal - UNIDERP, sob orientação do Prof. Msc. Carlos Cayres.

CAMPO GRANDE – MS

2011

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE CÓDIGOS-FONTE

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).

A mobilidade introduz problemas e desafios que não víamos, ou ignorávamos em ambientes fixos. Os problemas que a mobilidade impõe as redes de computador são os mais diversos e variados que se possa imaginar. Eles vão desde a velocidade do canal, passando por interferências do ambiente e localização da estação móvel, até duração da bateria desta estação. Estes e muitos outros parâmetros são fundamentais na hora de se projetar algoritmos para ambientes móveis, e que não nos preocupamos ao criar algoritmos baseados em redes fixas.

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 DISPOSITIVOS MÓVEIS

Pode-se dividir os dispositivos móveis para usuários em notebooks, *handhelts*, PDAs, e telefones celulares. Comumente, notebooks e *handhelts* são usados como computadores desktops, conectados a uma rede fixa e utilizando energia de “tomada”. Por outro lado, PDAs e celulares tendem a ser usados em situações que envolvem mobilidade, efetivamente fazendo uso de uma infraestrutura de comunicação sem fio e sem dependerem de energia de tomada. Também é comum encontrar dispositivos móveis projetados para aplicações específicas como a indústria. Atualmente, existe uma forte tendência em integrar PDAs e celulares. Assim, já é possível comprar PDAs com a função de celular e vice-versa. Com o surgimento de novas tecnologias de comunicação de dados, como a telefonia celular de terceira geração, novos dispositivos serão produzidos. É importante observar que a variação das interfaces de entrada e saída, capacidade de processamento e armazenamento,

autonomia de funcionamento, e suporte a protocolos de comunicação sem fio é bem grande entre os diferentes tipos de dispositivos. (Mateus & Ferreira Loureiro, 2004).

No ambiente de computação móvel, o tamanho do dispositivo é um fator muito importante, já que as pessoas não desejam carregar dispositivos pesados, mas sim leves e de fácil utilização. A tendência é que dispositivos móveis passem a ser usados mais frequentemente, como outros objetos que normalmente carregamos e passem a ser "incorporados" ao nosso dia-a-dia.

2.2 PERSPECTIVAS PARA APLICAÇÕES MÓVEIS

A chegada de novas infraestruturas de comunicação sem fio abrirá novas oportunidades para serviços de comunicação, entretenimento e gerenciamento, além de ampliar a disponibilização de serviços de conteúdo. O telefone celular e o PDA deverão continuar a evoluir para um terminal móvel com funcionalidades híbridas desses dois dispositivos. (Mateus & Ferreira Loureiro, 2004).

A utilização dessas tecnologias de forma integrada expandirá ainda mais as possibilidades de aplicações. A personalização será uma característica importante. Cada pessoa poderá utilizar um determinado serviço de acordo com suas necessidades ou preferências. Para isso, será necessário ter mecanismos sofisticados de adaptação e gerência desses serviços. Também será comum ter uma aplicação com diferentes aspectos como uma aplicação de *M-Commerce* que utiliza um serviço baseado na localização.

2.3 SUPORTE A MOBILIDADE NA REDE FIXA

Do ponto de vista da rede fixa, o protocolo IP é o protocolo que cria a "cola" para integração de serviços e aplicações entre a rede fixa e a rede sem fio. Mais ainda, o protocolo IP Móvel cria as condições para dar suporte à mobilidade de dispositivos móveis dentro da rede fixa. O IPv6 móvel (*Mobile IP*) foi projetado para criar efetivamente uma Internet ubíqua para a computação móvel. Dentre as características importantes do IPv6 móvel para a computação móvel estão o espaço de endereçamento de 128 bits, e a forma de ligação de dispositivos baseada no princípio *plug-and-play*. (Mateus & Ferreira Loureiro, 2004).

2.4 LOCALIZAÇÃO DE UNIDADES MÓVEIS

A evolução das redes de comunicação sem fio tem possibilitado a comunicação mais flexível para as pessoas e aumentado consideravelmente o acesso a redes de serviços. (Mateus & Ferreira Loureiro, 2004).

Com o desenvolvimento destas redes e o aumento da mobilidade surge uma nova necessidade que é acessar serviços dependentes da localização do usuário. Infelizmente, tecnologias de computação móvel atuais possuem problemas relacionados com a comunicação sem fio, portabilidade e mobilidade.

Serviços baseados na Localização ou LBS (*Location Based Services*) 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 um LBS 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, um LBS 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.

O objetivo deste tipo de serviço é fornecer aos usuários informação que seja dependente de sua localização, tais como serviços de emergência, informações de trafego, bancos, restaurantes entre outros.

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.

Uma medida importante nesse contexto é a qualidade de serviço a ser oferecido, que implica, dentre outras coisas em diferentes tipos de adaptação. O processo de adaptação pode ser feito através de políticas.

Os serviços baseados em localização estão ainda em desenvolvimento e existem muitas questões que necessitam ser consideradas. Os principais problemas a serem tratados

podem ser divididos em três áreas: rede, localização do dispositivo e as funcionalidades fornecidas por esses dispositivos. Redes internas e externas possuem características diferentes como largura de banda e área de cobertura, além de protocolos diferentes.

2.5 SEGURANÇA

A tecnologia móvel, que oferece acesso a informação em qualquer lugar e a qualquer momento, traz um desafio mais sério para as pessoas e organizações que é o risco da segurança. Segundo previsões de 2002 do *Gartner Group* [Gartner], em 2007, mais de 200 milhões de dispositivos móveis serão usados por pessoas em atividades relacionadas com negócios. Possivelmente, nesse cenário haverá muitos problemas de segurança envolvendo dados armazenados em handhelds e transmitidos via comunicação sem fio. Os mecanismos de segurança em rede terão que ser revistos para evitar ou diminuir um uso incorreto acidental ou intencional do processo e informação no ambiente de computação móvel, já que esses tipos de problema tendem a causar uma desconfiança maior nas entidades envolvidas e tecnologias usadas. (Mateus & Ferreira Loureiro, 2004).

2.6 TECNOLOGIAS DE COMUNICAÇÃO SEM FIO

Dentre as diversas infraestruturas de comunicação sem fio existentes, as mais utilizadas são a comunicação celular de segunda geração chamada de 2G (baseada nos padrões TDMA, CDMA e GSM), a geração 2,5 que é uma solução intermediária baseada em comunicação de pacotes, a 3G, que nos próximos anos promete velocidades na faixa de Mbps, redes locais sem fio baseadas no padrão IEEE 802.11, redes pessoais baseadas no padrão Bluetooth e RFID (*Radio Frequency Identification*). (Mateus & Ferreira Loureiro, 2004).

Algumas das tecnologias de comunicação sem fio utilizadas atualmente estão descritos a seguir e mostradas na figura 2.6.

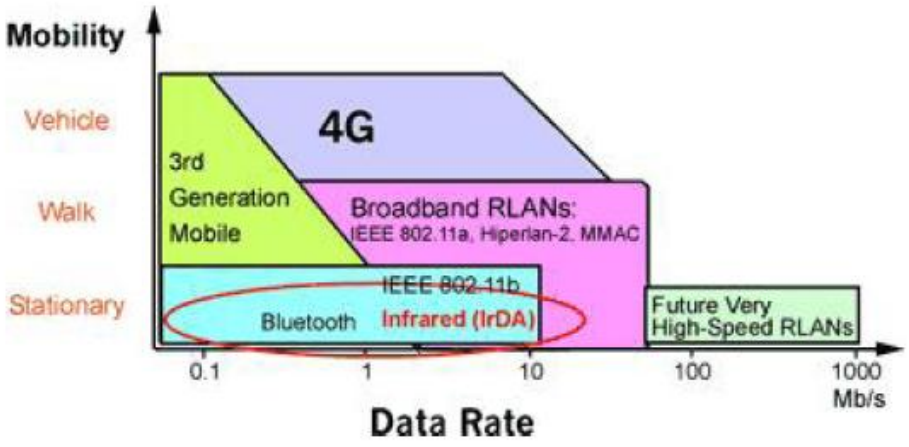

Figura 1 - Tecnologias de comunicação sem fio

2.6.1 WAP

O WAP é um método de distribuição de informação da Internet para os usuários, através de um dispositivo móvel que, atualmente, é padronizado pelo WAP Fórum. O modelo de programação WAP é similar ao modelo de programação Web. Isto significa que ele provê vários benefícios para a comunidade desenvolvedora de aplicações, incluindo um modelo de programação familiar e a capacidade de reutilização das ferramentas atuais, como os servidores Web. Entretanto, otimizações e extensões foram feitas de maneira que a característica do mundo Web fosse ao encontro do ambiente sem fio. Sempre que possível, os padrões existentes foram plenamente adotados ou foram usados como ponto de partida para a tecnologia WAP**.** (Mateus & Ferreira Loureiro, 2004).

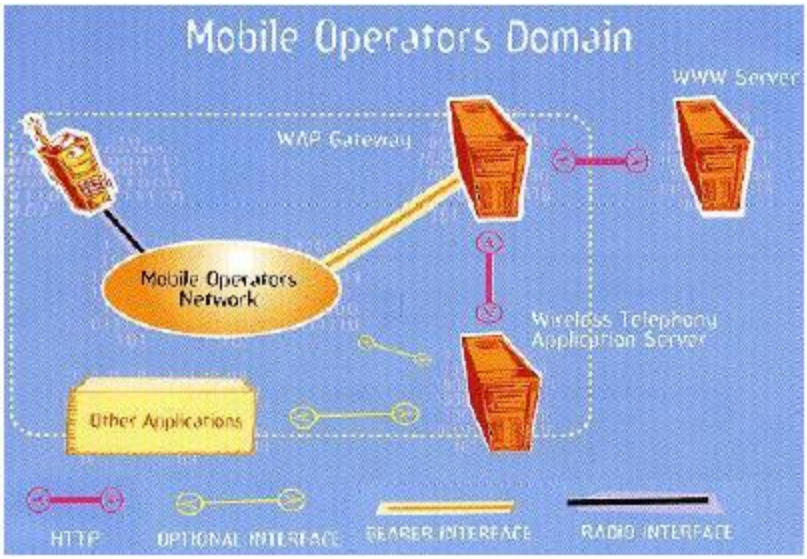

Figura 2 - Tecnologia WAP

2.6.2 I-MODE

O I-MODE foi criado pela operadora japonesa NTT DoCoMo [Interactive, 2000] e é basicamente um serviço de informação por pacotes. Com este sistema de informação "em pacotes", diferentemente das redes telefônicas de comutação de circuitos, não é necessário que cada usuário receba a informação através de um só canal de rádio, o que significa que um grande número de pessoas pode ter acesso à informação simultaneamente. Além disso, o modelo em pacotes ajuda a reduzir os custos, já que as tarifas baseiam-se no volume de informação enviada e recebida. (Mateus & Ferreira Loureiro, 2004).

O I-MODE se aproxima em muitos aspectos do WAP, a começar pela velocidade de transmissão, que é de 9,6 kbps, a mesma que se tem nas redes TDMA. Essa baixa taxa de transmissão faz com que o protocolo japonês esbarre na dificuldade de transportar imagens, assim como o WAP. São possíveis apenas ícones muito simples, parecidos com pequenas imagens do WAP em formato vbmp.

2.6.3 Bluetooth

Bluetooth é um padrão proposto pelo Bluetooth SIG (*Special Interest Group*) que é um consórcio das maiores empresas de telecomunicações e computação do mundo. O padrão opera na faixa ISM (*Industrial, Scientific, and Medical*) de 2,4 GHz e tem como princípio propor uma tecnologia de baixo custo para conectividade sem fio. Inicialmente o padrão foi projetado como uma solução para substituição de cabos usados na comunicação de periféricos por comunicação via rádio. No entanto, ele permite a conexão entre diferentes tipos de dispositivos possibilitando a formação de redes ad-hoc. (Mateus & Ferreira Loureiro, 2004).

2.6.4 IEEE 802.11

O padrão de comunicação IEEE 802.11 foi criado em 1999 para suportar a comunicação em Redes Locais Sem Fio, (WLANs – Wireless Local Networks). A especificação define uma camada de acesso ao meio, camada MAC, e diferentes camadas físicas. (Mateus & Ferreira Loureiro, 2004).

Atualmente, já é comum ter uma infraestrutura baseada no padrão 802.11 disponível para clientes em livrarias, cafeterias, e outros estabelecimentos comerciais.

2.6.5 Tecnologias 2G, 3G E 4G

Os sistemas móveis de terceira geração, chamados de sistemas IMT-2000, foram projetados para prover acesso a diferentes tipos de serviços de comunicação de dados, e também voz, dentre eles aplicações multimídia, acesso a Web e outras aplicações que precisam de uma largura de banda não encontrada normalmente em redes celulares 2G e 2,5G. (Mateus & Ferreira Loureiro, 2004).

No futuro, a tendência é que toda a infraestrutura de comunicação sem fio seja baseada numa rede comutada por pacotes, baseada no protocolo IP.

2.7 REDES ESTRUTURADAS

2.7.1 Redes de celulares

Os sistemas celulares são os mais populares sistemas sem fio. Muitos conceitos são particulares e outros extrapolam esta área. O nome sistema móvel celular (SMC) advém de sua estrutura em células. Uma célula é uma área geográfica atendida ou coberta por um transmissor de baixa potência, uma ERB (Estação Rádio Base), que é uma ou mais antenas fixas, instaladas em torres que têm como objetivo atender a demanda originada pelas estações ou unidades móveis, ou usuários, dentro de sua área de cobertura. (Mateus & Ferreira Loureiro, 2004).

A conexão entre uma ERB e uma unidade móvel se realiza por um canal ou frequência disponível. Cada ERB está conectada por uma linha física dedicada a uma CCC (Central de Comutação e Controle), que, por sua vez, também está conectada a RPT (Rede Pública de Telefonia). A CCC é responsável pela interligação e controle de várias ERBs.

2.7.2 Redes de satélites

Comunicações via satélite possuem características bastante peculiares, entre elas são a alta capacidade e possibilidade de atender um elevado número de usuários a baixo custo. A viabilidade econômica desses projetos se concentra no atendimento de massa global, a custos reduzidos (hoje são da ordem de 1 a 3 dólares/minuto), competitivos, sem fronteiras e, principalmente, complementando os serviços já existentes. (Mateus & Ferreira Loureiro, 2004).

Nesta linha, cobrem regiões não atendidas por sistemas terrestres, pela baixa densidade populacional, pela baixa renda, ou por dificuldades geográficas, caracterizando os seus maiores segmentos de comunicação sem fio fixo, de extensão celular e de internacionalização dos serviços celulares. Muitos projetos estão em andamento e têm sofrido muitos ajustes de objetivos, dimensões e implementações. Na concepção de mobilidade as células são unidades móveis enquanto os usuários estão fixos, devido ao posicionamento em altitudes elevadas. Os sinais transmitidos são recebidos por toda área coberta, uma ampla área geográfica, e o custo é independente da distância entre os usuários. Com isso, apresentam uma alta capacidade para transmissões broadcast e sistemas distribuídos.

3 MODELAGEM UML

A UML (*Unified Modeling Language* - Linguagem de Modelagem Unificada) tornou-se nos últimos anos a linguagem padrão de modelagem adotada internacionalmente pela indústria de Engenharia de Software. É uma linguagem visual utilizada para modelar sistemas computacionais por meio do paradigma de Orientação a Objetos. (Guedes, 2006).

No entanto a UML não é uma linguagem de programação e sim 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 UML surgiu da união de três metodologias de modelagem: o método de Booch, o método OMT (*Object Modeling technique*) de Jacobson e o método OOSE (*Object Oriented Software Engineering*) de Rumbaugh. Essas eram, até meados da década de 90, as três metodologias de modelagem orientada a objetos mais populares entre os profissionais da área de desenvolvimento de software. A união dessas metodologias contou com o amplo apoio da Rational Software, que incentivou e financiou a união das três metodologias.

O esforço inicial do projeto começou com a união do método de Booch com o método OMT de Jacobson, o que resultou no lançamento do Método Unificado no final de 1995. Logo em seguida, Rumbaugh juntou-se a Booch e Jacobson na Rational Software e seu método OOSE começou também a ser incorporado à nova metodologia. O trabalho de Booch, Jacobson e Rumbaugh, resultou no lançamento, em 1996, da primeira versão da UML propriamente dita.

Tão logo a primeira versão foi lançada, diversas grandes empresas atuantes na área de modelagem e desenvolvimento de software passaram a contribuir com o projeto, fornecendo sugestões para melhorar e ampliar a linguagem. Finalmente a UML foi adotada pela OMG (*Object Management Group* - grupo de gerenciamento de Objetos) em 1997, como uma linguagem padrão de modelagem.

Projetar uma linguagem a ser utilizada em analise e projeto orientados a objetos não é muito diferente de criar uma linguagem de programação. Primeiro, é necessário delimitar o

problema, onde a linguagem deverá abranger a especificação de requisitos, ser suficiente para permitir programação visual. Segundo, é preciso alcançar um equilíbrio entre expressividade e simplicidade. (Booch, Rumbaugh, & Jacobson, 2000).

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

A UML é uma linguagem padrão para a elaboração da estrutura de projetos de software, podendo ser empregada para a visualização, especificação, construção e a documentação de artefatos que façam uso de sistemas complexos de software. É uma linguagem muito expressiva, abrangendo todas as visões expressividade, não é difícil compreender e usar a UML. É apenas uma linguagem e, portanto, é somente uma parte de um método para desenvolvimento de software. É independente do processo, apesar de ser perfeitamente utilizada em processo orientado a casos de usos, centrado na arquitetura, iterativo e incremental.

Através da modelagem de um software, os profissionais conseguem determinar fatores extremamente complexos, como levantamento e analise de requisitos, prototipagem, tamanho do projeto, complexidade, prazos, custos, documentação, manutenção e reusabilidade entre outros. Por mais simples que seja, todo e qualquer sistema deve ser modelado antes de se

iniciar a implementação, entre outras coisas, pois os sistemas de informação frequentemente costumam possuir a propriedade de crescer, isto é, aumentar o tamanho, complexidade e abrangência. Sistemas de informação são dinâmicos, por que normalmente os sistemas de informação estão em constante mudança. (Guedes, 2006).

Para isso, um sistema de informação precisa possuir uma documentação extremamente detalhada, precisa e atualizada para que possa ser mantido com facilidade, rapidez e de maneira correta, sem produzir novos erros ao corrigir os antigos. Modelar um sistema é uma forma bastante eficiente de documentá-los, mas a modelagem não serve apenas para isso, a documentação é apenas uma das vantagens fornecidas pela modelagem.

3.1 LEVANTAMENTO DE REQUISITOS

Uma das primeiras fazes de engenharia de um software consiste no levantamento de requisitos. Nesta etapa, o engenheiro de software busca compreender as necessidades do usuário e o que deseja que o sistema a ser desenvolvido realize. Isto é feito principalmente por meio de entrevistas, onde o engenheiro de software tenta compreender como funciona atualmente o processo a ser informatizado e quais serviços o cliente precisa que o software forneça. (Guedes, 2006).

3.2 ANÁLISE DE REQUISITOS

Logos após o levantamento de requisitos passa-se à fase em que as necessidades apresentadas pelo cliente são analisadas, esta etapa é conhecida como analise de requisitos, onde o engenheiro examina os requisitos enunciados pelos usuários, verificando se estes foram especificados corretamente e se foram realmente bem compreendidos. A partir da etapa de analise de requisitos são determinadas as reais necessidades do sistema de informação.

3.3 PROTOTIPAGEM

É uma técnica bastante popular e de fácil aplicação. Essa técnica consiste em desenvolver rapidamente em rascunho do que seria o sistema de informação quando este estivesse finalizado. Um protótipo normalmente apresenta pouco mais do que a interface do software a ser desenvolvido, ilustrando como as informações seriam inseridas e recuperadas

no sistema e apresentando alguns exemplos com dados fictícios de quais os resultados apresentados pelo software, principalmente em forma de relatórios. A utilização de um protótipo pose assim evitar que após meses ou mesmo anos de desenvolvimento, descubra-se, ao implantar o sistema, que o software não atende completamente as necessidades do cliente devido principalmente a falhas de comunicação durante as entrevistas iniciais.

3.4 MANUTENCAO

Possivelmente mais importante que todas as outras questões já enunciadas anteriormente, existe a questão da manutenção. Muitas vezes a manutenção de um software pode representar de 40 a 60% do custo total do projeto. Pode-se dizer que a modelagem é necessária para diminuir os custos com a manutenção, se a modelagem estiver correta, assim o sistema não apresentará erros e então não precisará sofrer manutenções.

É considerada uma tarefa ingrata pelos profissionais de desenvolvimento, por normalmente exigir que estes despendam grandes esforços para compreender códigos escritos por outros profissionais com estilos de desenvolvimento diferentes e que normalmente não se encontram mais na empresa.

3.5 DOCUMENTCAO

É por meio da documentação histórica que a empresa pode saber se está evoluindo, se o processo de desenvolvimento está se tornando mais ágil, se as metodologias utilizadas atualmente são superiores as anteriores e se a qualidade do software produzido está melhorando.

Uma empresa ou setor de desenvolvimento de software necessita de um registro detalhado de cada um de seus sistemas de informação anteriormente desenvolvidos, para poder determinar o tempo gasto na manutenção de um sistema, a média de custos com a modelagem e desenvolvimento e a quantidade de profissionais necessários para o desenvolvimento de um tipo de sistema.

Além disso, a documentação pode ser muito útil na reusabilidade de um sistema ou parte de seu código-fonte.

3.6 CLASSES

São os blocos de construção mais importantes de qualquer sistema orientado a objetos. Uma classe é uma descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica. Uma classe implementa uma ou mais interfaces. Podem incluir abstrações que são parte do domínio do problema, assim como as classes que fazem uma implementação, podem representar itens de software, de hardware e ate mais itens que sejam puramente conceituais. Classes bem estruturadas anulam fronteiras e formam uma parte de uma distribuição equilibrada de responsabilidade em um sistema. (Booch, Rumbaugh, & Jacobson, 2000).

3.7 RELACIONAMENTOS

A maioria das classes trabalham em colaboração uma com as outras de varias maneiras, formando assim um relacionamento. Existem três tipos de relacionamentos especialmente importantes:

- Dependências, que representam relacionamentos de utilização entre as classes, como refinamento, rastreamento e vínculos;
- Generalizações, que relacionam classes generalizadas a suas especializações.
- Associações, que representam relacionamentos estruturais entre objetos.

Cada um desses relacionamentos fornece uma forma diferente de combinações de abstrações.

3.8 MECANISMOS

A UML se torna mais simples devido à presença de quatro mecanismos básicos que são aplicados de maneira consistente na linguagem: especificações, adornos, divisões comuns, e mecanismos de extensibilidade.

As notas são o tipo mais importante de adorno que pode aparecer sozinho. Uma nota é um símbolo gráfico para a representação de restrições ou de comentários anexados a um elemento ou a uma coleção de elementos. Use as notas para anexar informações a um modelo, como requisitos, observações, revisões e explicações. Os mecanismos de extensibilidade da

UML permitem estender a linguagem de uma maneira controlada. Esses mecanismos incluem estereótipos, valores atribuídos e restrições. Os estereótipos ampliam o vocabulário da UML, permitindo a criação de novos tipos de blocos de construção, derivados de outros já existentes, mas específicos a determinado problema, um valor atribuído estende as propriedades de blocos de construção da UML, permitindo a criação de novas informações nas especificações desses elementos. Uma restrição estende a semântica dos blocos de construção da UML, permitindo adicionar novas regras ou modificar as existentes. Uses esses mecanismos para ajudar a UML às necessidades especificas do seu domínio e cultura de desenvolvimento.

3.9 DIAGRAMAS

Ao fazer uma modelagem, se cria uma simplificação da realidade para entender melhor o sistema em desenvolvimento. Usando a UML, se constrói seus modelos a partir de blocos de construção básicos, como classes, interfaces, colaborações, componentes, nós, dependências, generalizações e associações.

Diagramas são meios utilizados para a visualização desses blocos de construção. Um diagrama é uma apresentação gráfica de um conjunto de elementos, geralmente representados como um gráfico conectado de vértices (itens) e arcos (relacionamentos). Usa-se os diagramas para visualizar o sistema sob diferentes perspectivas. Uma vez que nenhum sistema complexo pode ser compreendido em sua totalidade a partir de uma única perspectiva, a UML define um numero de diagramas que permite dirigir o foco para aspectos diferentes do sistema de maneira independente.

3.9.1 Diagramas de Casos de Uso

O diagrama de caso de uso é o diagrama mais geral e informal da UML, sendo utilizado normalmente nas fases de levantamento e analise de requisitos do sistema, embora venha a ser consultado durante todo o processo de modelagem e possa servir de base para outros diagramas, apresenta uma linguagem simples e de fácil compreensão para que os usuários possam ter uma ideia geral de como o sistema irá se comportar. Procura identificar os atores (usuários, outros sistemas ou ate mesmo algum hardware especial) que utilizarão de alguma forma o software, bem como os serviços, ou seja, as opções, que o sistema

disponibilizará aos autores, conhecidas neste diagrama como casos de uso, como mostrado na figura abaixo:

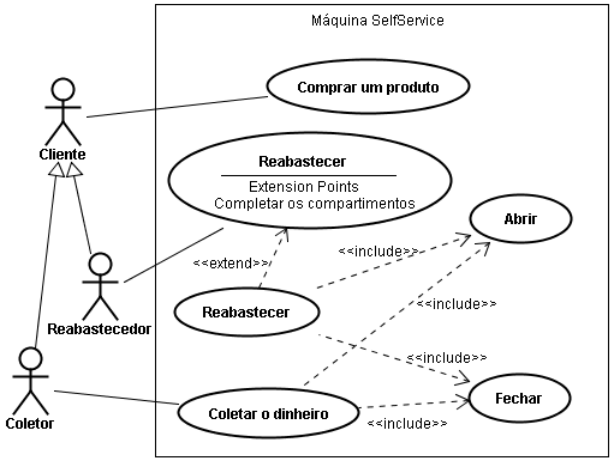

Figura 3 - Diagrama de Caso de Uso

3.9.2 Diagramas de Classes

É o diagrama mais utilizado e o mais importante da UML, servindo de apoio para a maioria dos outros diagramas. Como o próprio nome diz, define a estrutura das classes utilizadas pelo sistema, determinando os atributos e métodos possuídos por cada classe, além de estabelecer como as classes se relacionam e trocam informações entre si. Veja a figura abaixo um exemplo de diagrama de classe: (Guedes, 2006).

São os diagramas encontrados com maior frequência na modelagem de sistemas orientados a objetos. Um diagrama de classes mostra um conjunto de classes, interfaces e colaborações e seus relacionamentos.

Usa-se diagramas de classes para fazer a modelagem da visão estática do projeto de um sistema. Na maioria dos casos, isso envolve a modelagem do vocabulário do sistema, a modelagem de colaborações ou a modelagem de esquemas. Os diagramas de classes também são a base para um par de diagramas relacionados: os diagramas de componentes e os diagramas de implantação.

Os diagramas de classes são importantes não só para a visualização, a especificação e a documentação de modelos estruturais, mas também para a construção de sistemas executáveis por intermédio de engenharia de produção e reserva. (Booch, Rumbaugh, & Jacobson, 2000).

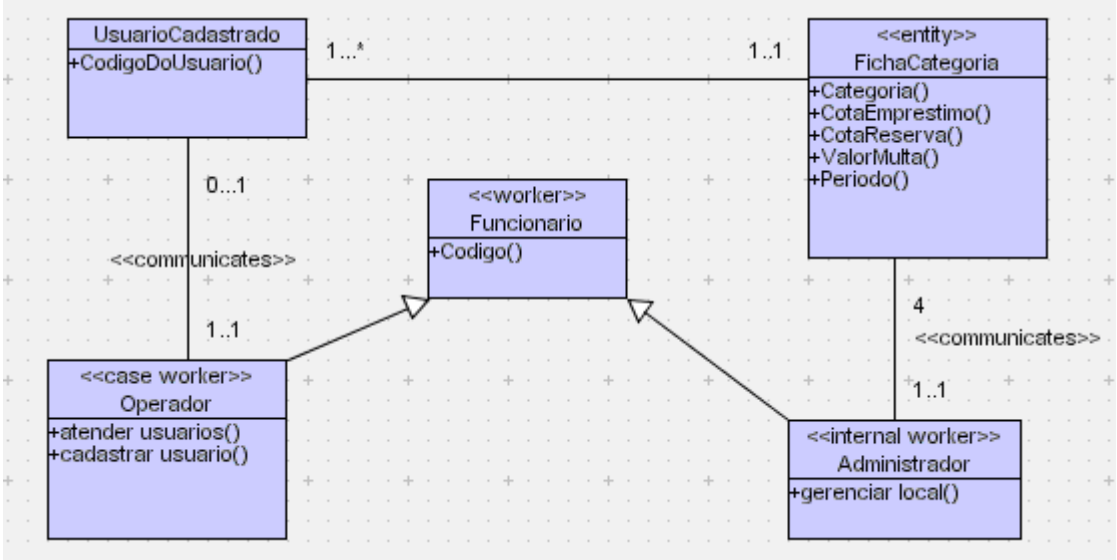

Figura 4 - Diagrama de Classes

3.9.3 Diagramas de Sequência

Este diagrama preocupa-se com a ordem temporal em que as mensagens são trocadas entre os objetos envolvidos em um determinado processo. Em geral, baseia-se em um caso de uso definido pelo diagrama de mesmo nome e apoia-se no diagrama de classes para determinar os objetos das classes envolvidas em um processo. Um diagrama de sequência costuma identificar o evento gerador do processo modelado, bem como o ator responsável por este evento e determina como o processo deve se desenrolar e ser concluído por meio da chamada de métodos disparados por mensagens enviadas entre os objetos, como mostrado na figura abaixo:

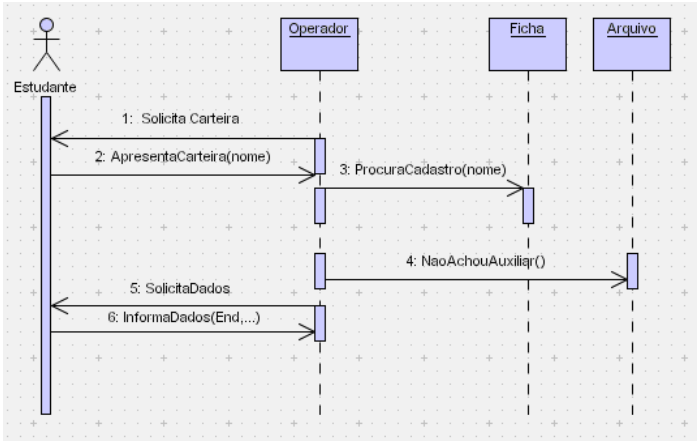

Figura 5 - Diagrama de Sequência

3.9.4 Diagramas de Implantação

Determina as necessidades de hardware do sistema, as características físicas como servidores, estações, topologias e protocolos de comunicação, ou seja, todo o aparato físico sobre o qual o sistema deverá ser executado. A figura abaixo apresenta um exemplo desse diagrama:

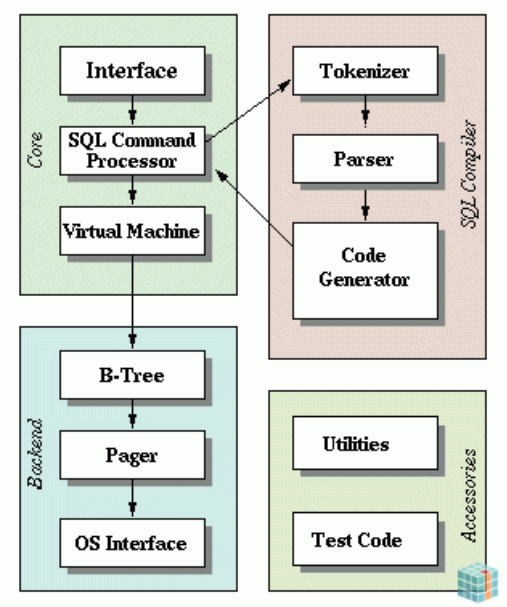

Figura 6 - Diagrama de Implantação

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:

1. 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.
2. 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.
3. 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.

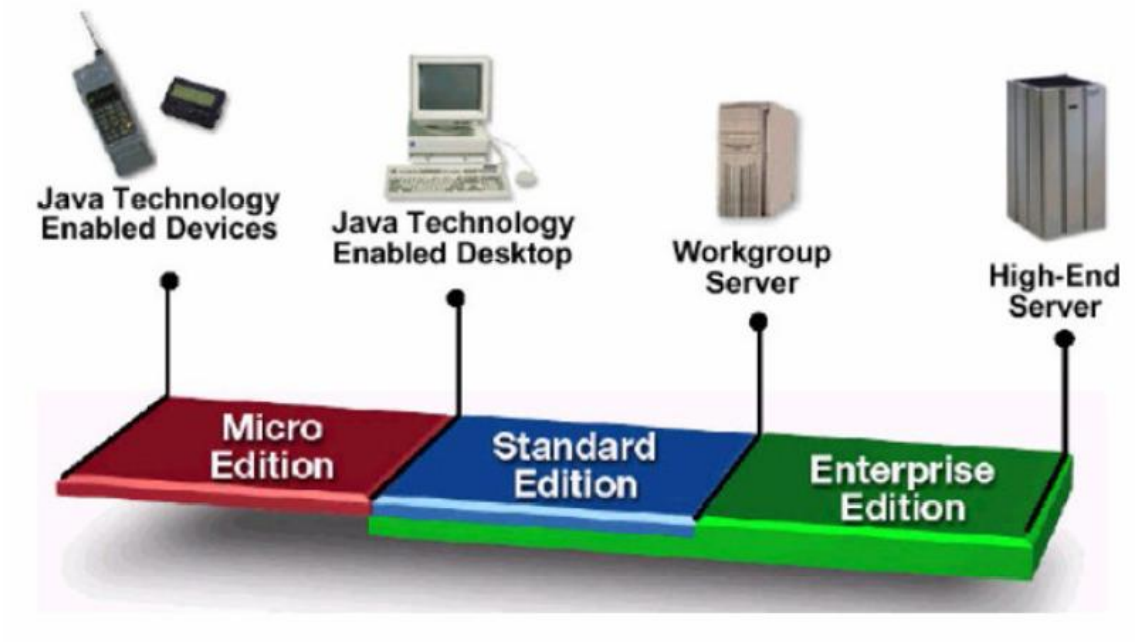

Figura 7 - Edições da Plataforma JAVA

4.1 JAVA MICRO EDITION (JME)

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.

4.1.1 Configuração

A *Configuration* define o mínimo que um desenvolvedor pode esperar de um dispositivo, classificando-os por capacidade de memória e processamento. Especifica uma JVM que pode ser portada entre dispositivos e também determina um subconjunto das APIs da JSE a serem disponibilizadas, além de outras APIs adicionais necessárias. Entre as configurações disponíveis destacam-se a CDC (*Connected Device Configuration*) e a CLDC (*Connected, Limited Device Configuration*). (Kliemann, 2006).

A especificação CDC é baseada na máquina virtual Java convencional, definindo um ambiente com um conjunto relativamente rico de recursos, muito semelhante ao encontrado em um sistema desktop. Destina-se a dispositivos *wireless* (sem fio) de alta capacidade, *set-top boxes* de TVs a cabo, sistemas automotivos e outras plataformas que possuam pelo uma quantidade relativamente alta de megabytes de memória disponível. A CLDC, por outro lado, consiste em uma máquina virtual reduzida (KVM) e um conjunto de classes mais apropriado para dispositivos pequenos e com limitações significativas de desempenho e memória. Essa configuração é destinada para dispositivos *wireless* (sem fio) menores, possuindo geralmente entre 160kb e 10MB (no mínimo 128 Kb) de memória, uma conexão à rede limitada, intermitente e provavelmente lenta. A tela é de tamanho pequeno, e a fonte de energia é reduzida, fornecida por baterias. Tal ambiente é direcionado para telefones celulares, Pager, PDAs e outros.

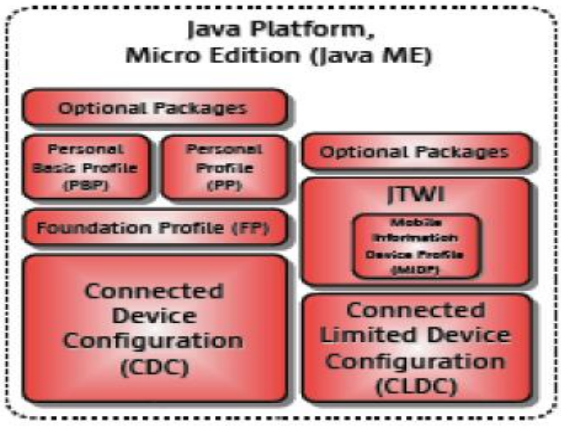

Figura 8 - Configurações JME

4.1.2 Profiles

Os *Profiles* (perfis) são conjuntos de APIs que suplementam as *Configurations*, fornecendo funcionalidades para um determinado tipo de dispositivo. São mais específicos que as configurações, apesar de serem baseados nelas. Adicionam APIs para interface com o

usuário, armazenamento persistente e outras auxiliares. O *profile* utilizado nos celulares é o MIDP (*Mobile Information Device Profile*), com o *Foundation Profile* sendo utilizado em dispositivos de rede sem interface gráfica, e o *Personal Basis* e *Personal Profile* utilizados em dispositivos com suporte gráfico e alta capacidade sobre a CDC. (Kliemann, 2006).

Dentre as características do MIDP 1.0, temos as exigências mínimas de 128 Kb de memória não volátil para a implementação MIDP, 32 Kb de memória volátil para o *heap* em *runtime* (memória volátil em tempo em execução), 8 Kb de memória não volátil para persistência de dados, uma tela com pelo menos 96 x 54 *pixels*, algum tipo de entrada de dados (seja por teclado, teclas de telefone ou *touch screen*), e conexões de rede (*two-way*), possivelmente intermitentes. A segurança das aplicações Java em MIDP 1.0 é semelhante às *applets*, com o conceito de uma *sandbox*, onde o programa Java é mantido seguro, sem acesso ao ambiente externo.

A MIDP 2.0 é uma versão revisada da versão 1.0. Novos recursos adicionam uma melhor interface de usuário, multimídia e funcionalidades de jogos, conectividade mais ampla, abastecimento *over-the-air* (OTA), auto acionamento de aplicativos (PushRegistry) e segurança *end-to-end* (entre fonte e destino).

4.1.3 Pacotes Opcionais

As APIs opcionais são funcionalidades adicionais específicas que não serão encontradas em todos os dispositivos de uma determinada configuração ou perfil, mas importantes o suficiente para serem padronizadas. Criadas para disponibilizar requisitos muito específicos de aplicações, esses pacotes opcionais oferecem APIs padrão para uso em ambas as tecnologias existentes e emergentes como conectividade à base de dados, mensagens sem fio, multimídia, gráficos 3D, *Web Services* e Localização. Pelo fato dos pacotes opcionais serem modulares, fabricantes de dispositivos evitam sobrecarregar os dispositivos de funcionalidades desnecessárias instalando apenas os pacotes que as aplicações realmente irão utilizar. Os pacotes opcionais podem ser implementados lado a lado virtualmente a qualquer combinação de configurações e perfis. (Kliemann, 2006).

As especificações da plataforma J2ME, assim como também ocorre para as edições J2SE e J2EE da plataforma Java, são desenvolvidas sob o amparo da *Java Community Process* (JCP). Uma especificação Java inicia sua vida como uma requisição de especificação, *Java Specification Request* (JSR). Um grupo experiente consistindo de representantes de

companhias participantes é formado para criar a especificação. A JSR passa então por diversos estágios no JCP antes de ser finalizada. Toda JSR recebe um número. Especificações J2ME são comumente referenciadas pelas suas JSRs. Por exemplo, a JSR179 corresponde a API Location do J2ME.

No modelo de solução apresentado nesse trabalho utilizaremos algumas APIs opcionais além de outros recursos do J2ME. Os de maior destaque são: *PushRegistry* (recurso disponível no novo perfil MIDP 2.0) e *Location* API (JSR-179). Outras APIs do J2ME de menor relevância utilizadas no modelo e outras APIs possíveis de serem utilizadas não serão discriminadas.

4.1.4 APIs

Aplicações J2ME são desenvolvidas em um ambiente de programação e simulação qualquer. Muitos destes são gratuitos podendo ser carregados diretamente da página do fabricante. Juntamente com estes ambientes de desenvolvimento é fornecida a API de base onde estão presentes as classes de um perfil e de uma configuração. Na maioria dos casos o fabricante oferece um conjunto otimizado de classes para um determinado produto que endereçam aspectos mais específicos de hardware. (Kliemann, 2006).

Há hoje um número de JSRs relacionadas com a tecnologia Java ME, e este número está a aumentar rapidamente. Na família da tecnologia Java ME há uma série de JSRs diferentes com partes da plataforma e componentes definidos. (*Sun Microsystems*).

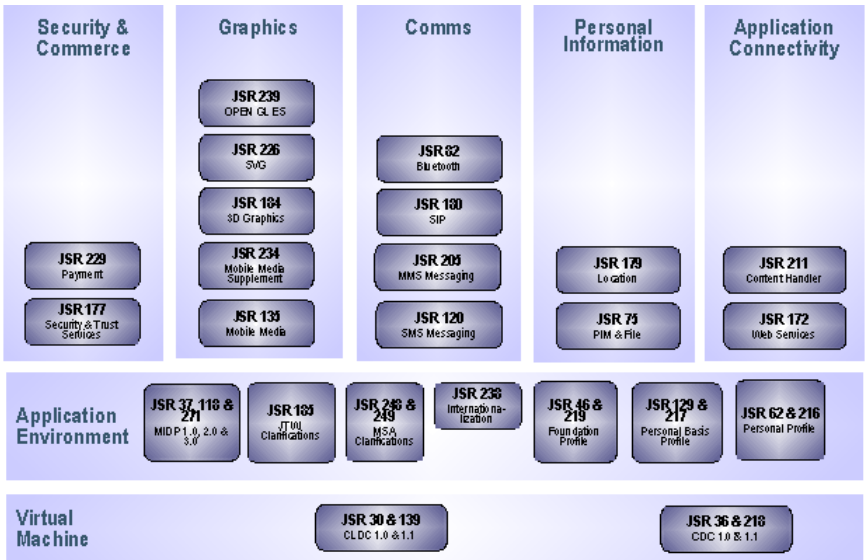

Figura 9 - JSRs relacionadas com Tecnologia

- ***Mobile Information Device Profile 2.0 (JSR 118)***

Aprimorou a parte de jogos com as classes *Sprite*, *TiledLayer* etc. (Silva, 2007).

- ***Mobile Media API (JSR 135)***

Habilidade para executar/gravar arquivos de mídia (áudio/vídeo)

Suporte a vários codecs

- ***Mobile 3D Graphics API for J2ME™ (JSR 184)***

Projetado de forma parecida a API do Java 3D™

Baseado no OpenGL–ES

- ***Messaging API (JSR 205)***

SMS (Texto) envio e recebimento de mensagem

MMS (Multimídia) envio e recebimento de mensagem

- ***File and PIM (JSR 75)***

Acesso ao sistema de arquivos do aparelho

Suporte a mídia removível, como os cartões de memória.

Acesso ao calendário

Acesso a lista de contatos

- ***Bluetooth (JSR 82)***

Bluetooth serviço/dispositivo descoberta e comunicação

- ***Location (JSR 179)***

Descobrir a localização atual

Cálculo da distância entre dois lugares, etc.

Diferentes métodos de localização suportados; por exemplo,

GPS interno ou GPS externo

- ***Session Initiation Protocol (JSR 180)***

  Permite o envio e recebimento de mensagens SIP

  Comunicação P2P sobre a rede

- ***Vector Graphics (JSR 226)***

  Visualização de dados de mapas

- ***Internationalization (JSR 238)***

  Localizar a aplicação

- ***Web Services (JSR 172)***

  Acesso e tratamento de dados

- ***Messaging (JSR 205)***

  Envio de imagens, vídeo, áudio e/ou texto para os amigos.

- ***Multimedia (JSR 234)***

  Captura de imagem, vídeo e áudio.

  Execução de áudio/vídeo

4.1.5 Exemplo de JME

Para exemplificar a implementação usando JME, demonstra-se um programa na listagem 1, que define dois objetos, um para validação de comando e outro caixa de texto.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class HelloWorld extends MIDlet implements CommandListener {
    private Command exitCommand;
    private TextBox tbox;

    public HelloWorld() {
        exitCommand = new Command("Exit", Command.EXIT, 1);
        tbox = new TextBox("Hello world MIDlet", "Hello World!", 25, 0);
        tbox.addCommand(exitCommand);
        tbox.setCommandListener(this);
    }

    protected void startApp() {
        Display.getDisplay(this).setCurrent(tbox);
    }

    protected void pauseApp() {}
    protected void destroyApp(boolean bool) {}

    public void commandAction(Command cmd, Displayable disp) {
        if (cmd == exitCommand) {
            destroyApp(false);
            notifyDestroyed();
        }
    }
}
```

Figura 10: Exemplo Hello World em JME

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 11.

Figura 11: 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)..

4.2.1 Atividades

São comparáveis aos utilitários independentes em sistemas desktop, tais como aplicações do tipo office. Atividades são trechos de código executável que vão e voltam no tempo, instanciadas pelo usuário ou pelo sistema operacional e executadas enquanto forem necessárias. Elas podem interagir com usuários e solicitar dados ou serviços de outras atividades ou serviços por meios de consultas ou intenções. (Rogers, Lombardo, Mednieks, & Meike, 2009).

A maior parte do código executável criado para o Android será executada no contexto de uma atividade. As atividades normalmente correspondem a telas de exibição, cada atividade exibe uma tela para o usuário. Quando não estiver em execução ativa, uma atividade pode ser eliminada pelo sistema operacional para economizar memória. (Rogers, Lombardo, Mednieks, & Meike, 2009).

4.2.2 Serviços

São semelhantes aos serviços nos sistemas operacionais de servidor ou desktop. São trechos executáveis de código, que geralmente são executados em segundo plano a partir do momento de sua instanciação ate o desligamento do dispositivo móvel. Geralmente não exibem interface de usuário. (Rogers, Lombardo, Mednieks, & Meike, 2009).

O exemplo clássico de um serviço é um MP3 player que precisa continuar reproduzindo uma fila de arquivos, mesmo quando o usuário passou a usar outras aplicações. Sua aplicação talvez precise implementar serviços para executar tarefas em segundo plano que persistam sem uma interface de usuário. (Rogers, Lombardo, Mednieks, & Meike, 2009).

4.2.3 Receptores de Broadcast e Intenção

Estes respondem às solicitações de serviço de outra aplicação. Um Receptor de broadcast responde a um aviso global de evento. Esses avisos podem ter origem no próprio Android ou qualquer programa em execução no sistema. Uma atividade ou serviço fornece a outras aplicações o acesso à sua funcionalidade pela execução de um receptor de Intenção, um pequeno trecho de código executável que responde a solicitações de dados ou serviços de

outras atividades. A atividade solicitante envia uma intenção, deixando ao framework do Android a decisão de qual aplicação deve receber a solicitação e agir conforme necessário. (Rogers, Lombardo, Mednieks, & Meike, 2009).

Intenções são elementos essenciais da arquitetura do Android que facilitam a criação de novas aplicações a partir das existentes. São usadas para a interação com outras aplicações e serviços que forneçam informações necessárias à sua aplicação. (Rogers, Lombardo, Mednieks, & Meike, 2009).

4.2.4 Provedores de conteúdo

São criados para compartilhar dados com outras atividades ou serviços. Um provedor de conteúdo usa uma interface padrão sob a forma de um URI para atender solicitações de dados de outras aplicações que talvez ainda não saibam qual provedor de conteúdo estão usando. (Rogers, Lombardo, Mednieks, & Meike, 2009).

O sistema operacional verifica quais aplicações se registraram como provedores de conteúdo para determinado URI e envia a solicitação à aplicação adequada (iniciando a aplicação se ela ainda não estiver em execução). Se houver mais de um provedor de conteúdo registrado para o URI solicitado o sistema operacional pergunta ao usuário qual ele pretende utilizar. (Rogers, Lombardo, Mednieks, & Meike, 2009).

Uma aplicação não precisa usar todos os componentes do Android, mas uma aplicação bem elaborada empregará os mecanismos fornecidos, em vez de reinventar a funcionalidade. A combinação de URIs e Intenções permite ao Android fornecer um ambiente de usuário muito flexível, as aplicações podem ser facilmente adicionadas, excluídas e substituídas, e o fraco acoplamento entre intenções e URIs mantem um bom funcionamento conjunto. (Rogers, Lombardo, Mednieks, & Meike, 2009).

Cada pacote *.apk* contém um arquivo de manifesto onde estão declarados todos os componentes da aplicação. O arquivo é estruturado em linguagem XML e também é utilizado para declarar as bibliotecas utilizadas, permissões, versão e requisitos. (Martins, 2009).

Quando o primeiro componente de uma aplicação precisa ser executado, um processo com um único thread é iniciado. Por padrão, todos os componentes da aplicação são executados neste thread. O Android dispõe de um mecanismo que permite a chamada de procedimentos remotos, executados em outros processos. (Martins, 2009).

Em determinado momento, quando a memória está escassa, por exemplo, o Android pode destruir um processo. Consequentemente, todos os componentes da aplicação que estão sendo executados naquele processo são destruídos. Para decidir qual processo deve ser eliminado, é levada em conta a importância dos processos para o usuário. Por exemplo, o sistema irá destruir primeiro aqueles que contêm atividades que não estão mais visíveis na tela. (Martins, 2009).

Cada componente de uma aplicação tem um ciclo de vida bem definido que inicia no momento em que é instanciado e acaba quando é destruído. Enquanto o componente está sendo executado, seu estado pode ser alterado diversas vezes.

Uma atividade possui essencialmente três estados: ativo, pausado e parado. As transições entre estes estados são notificadas através de chamadas a sete métodos especiais, conforme demonstrado na Figura 4. Estes métodos são utilizados para realizar as ações apropriadas a uma determinada mudança de estado. (Martins, 2009).

Diferentemente de sistemas operacionais para desktop, que geralmente disponibilizam um sistema de arquivos comum, no Android todos os dados são visíveis apenas para a aplicação dona. Para que estas informações sejam acessadas por outras aplicações, deve ser utilizado um componente do tipo provedor de conteúdo. (Martins, 2009).

O Android permite que os dados sejam armazenados de diversas formas, como através de um mecanismo chamado de preferências, onde é possível armazenar tipos primitivos. Esta possibilidade geralmente é utilizada para guardar as preferências do usuário. (Martins, 2009).

Também é possível armazenar dados diretamente no aparelho ou em um dispositivo de memória removível, utilizando arquivos. A aplicação pode, alternativamente, com o Sqlite, armazenar informações em tabelas em um banco de dados. A plataforma suporta ainda o acesso a operações de rede que podem ser utilizadas para guardar ou requisitar dados. (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.

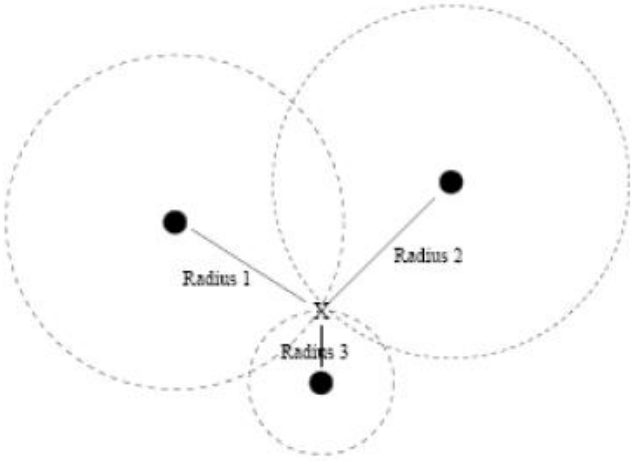

Figura 12 - Método de Triangulação

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:

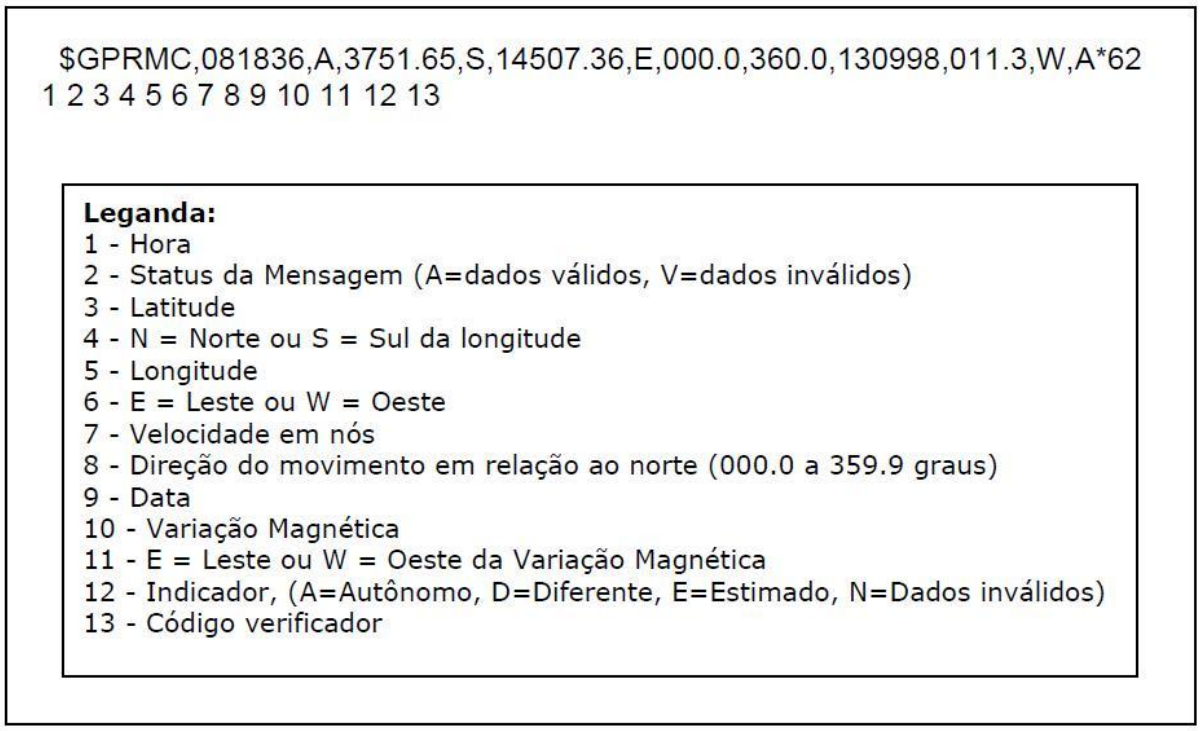

Figura 13 - Mensagem no padrão NMEA-0183 em formato GPRMC

5.2 CARACTERÍSTICAS

5.2.1 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).

5.2.2 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.

5.2.3 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).

5.3 APLICAÇÕES QUE PODEM USAR GPS

Podemos observar que todas estas definições têm em comum o fator localização. Com efeito, os LBS não são mais do que um serviço baseado na localização atual do usuário. A iniciativa do serviço prestado pode partir do usuário, por exemplo, saber a localização de um dado tipo de restaurante, e o modo de como chegar lá, ou então pode partir da parte do servidor do LBS, por exemplo, ao detectar a proximidade do usuário de um dado restaurante que lançará uma promoção, o LBS gera de imediato essa mensagem. (Kliemann, 2006).

Podemos classificar esses tipos de serviços em seis grandes categorias principais

- Gestão e coordenação de tráfego e rastreamento
- Envio de conteúdos e mensagens publicitárias
- Serviços turísticos integrados
- Serviços de Saúde
- Entretenimento
- Sistemas Móveis de Informações Ambientais

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.

6.1.1 Diagrama de caso de uso

A **figura 13** exibe o diagrama de caso de uso do desenvolvimento do protótipo proposto.

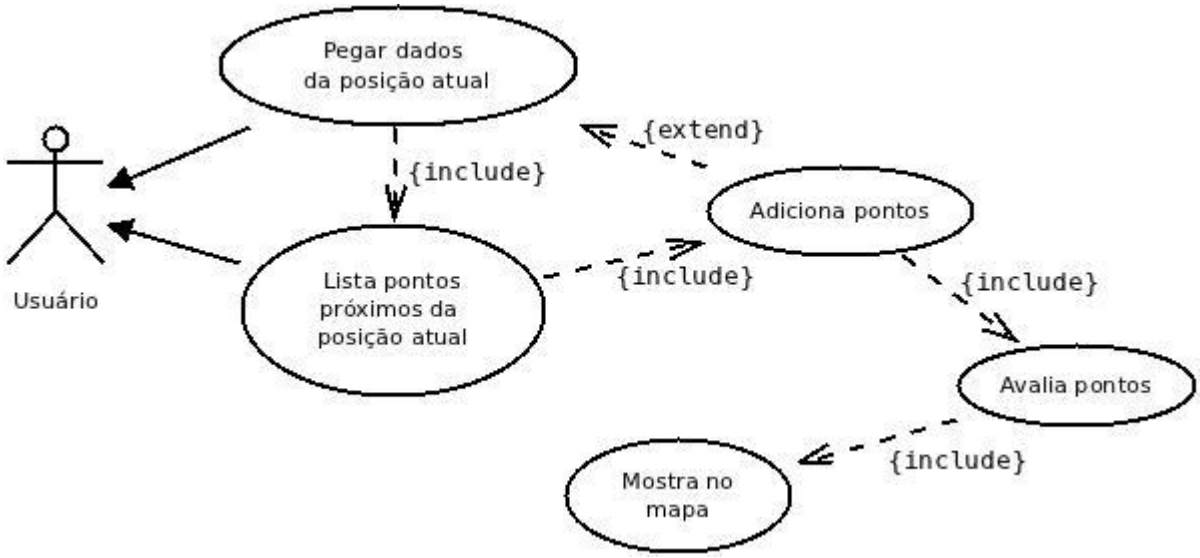

Figura 14: Diagrama de Caso de Uso do Protótipo SMBL

6.1.2 Diagrama de classe

A **figura 14** ilustra o diagrama de classe do protótipo Gasosa com suas respectivas classes.

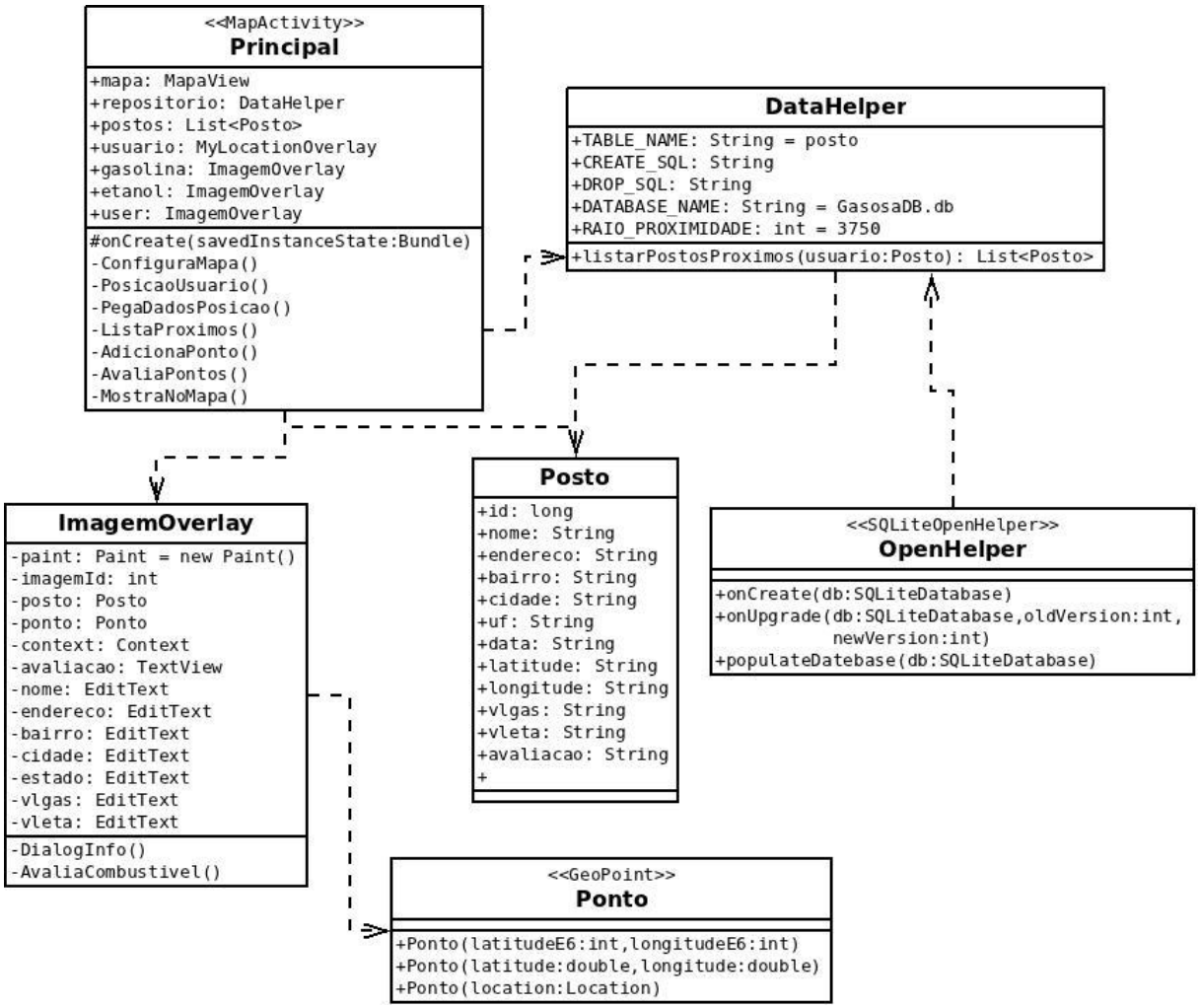

Figura 15: Diagrama de Classe do protótipo SMBL

6.1.3 Fluxograma

Conforme mostrado na figura 15 abaixo, vemos o fluxograma do protótipo Gasosa.

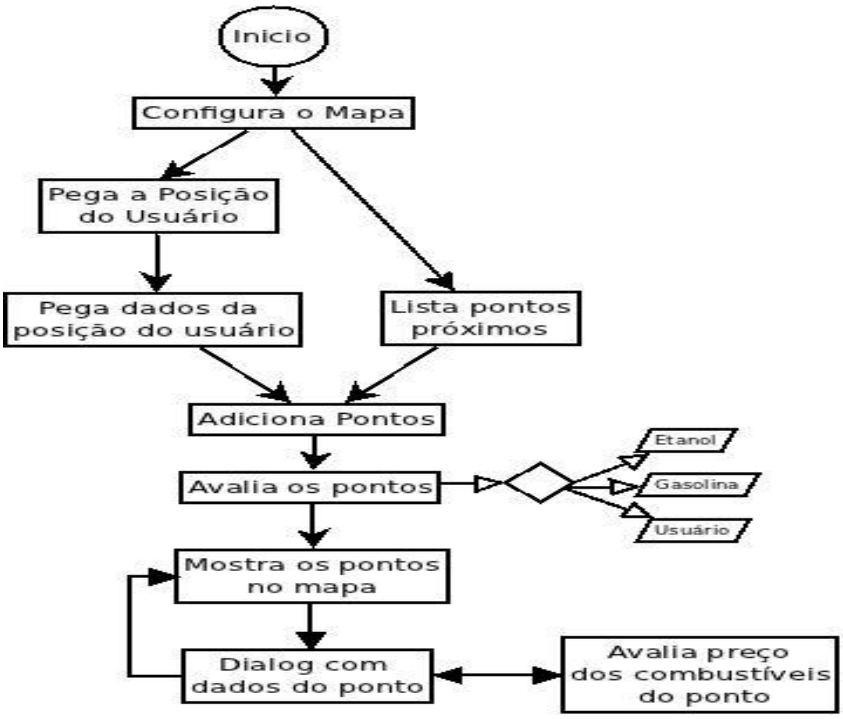

Figura 16: Diagrama de sequencia do protótipo SMBL

6.2 CODIFICAÇÃO DO PROTOTIPO

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

6.2.1 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*

6.2.2 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

6.2.3 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 5: 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 + " - Endereco: " + 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 6: 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 7: 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 8: Método *AvaliaCombustivel*

6.2.4 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 9: 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 10: 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 11: Método *populateDatabase*

6.2.5 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 12: Método *Posto*

6.2.6 **Classe *Ponto*:** Classe responsável pela conversão das coordenadas.

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

```
14    public Ponto(double latitude, double longitude){
15        this((int)(latitude * 1E6), (int)(longitude
16    }
```

Listagem 13: 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.

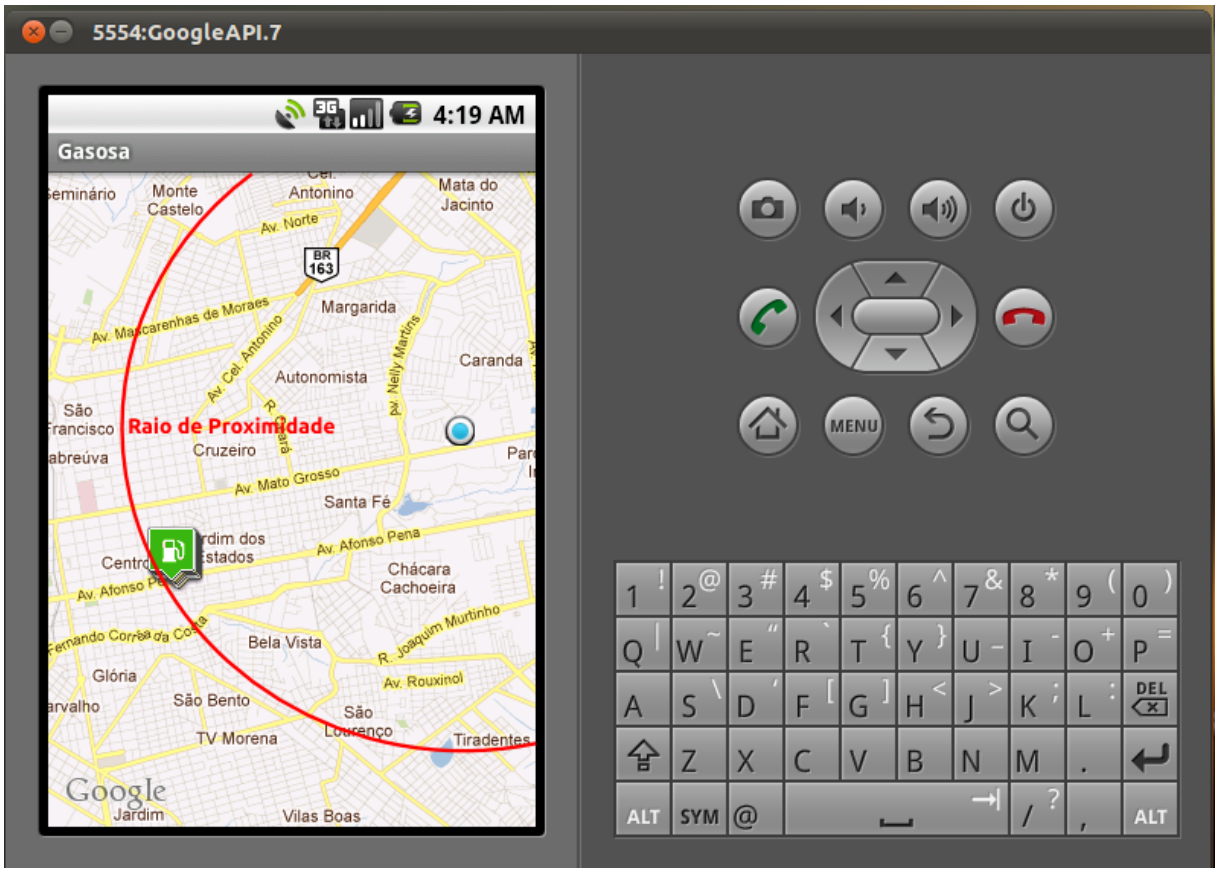

Figura 17: Tela *Principal*

- **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 18: 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 19: Tela *DialogInfo* exibindo o resultado de viabilidade

7 CONCLUSÃO

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.

8 REFERÊNCIAS BIBLIOGRÁFICAS

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.