Find* Parking Place: Um Sistema Ubíquo de Gerência para Estacionamentos

Ramon Mota Luz
Orientadora – Flávia Maristela S. Nascimento
Grupo de Pesquisa em Sistemas Distribuídos, Otimização, Redes e Tempo-Real (GSORT)
Especialização em Computação Distribuída e Ubíqua (ECDU)
Instituto Federal da Bahia (IFBA)
Email: ramonluz@ifba.edu.br

***Resumo*—A miniaturização dos dispositivos junto ao surgimento das redes sem fio serviram de alicerce para o crescimento das pesquisas em computação pervasiva e ubíqua. Neste contexto, diversas aplicações vem sendo desenvolvidas com foco em dar suporte as diversas atividades cotidianas. Um dos problemas que tem ganhado destaque no cenário dos grandes centros urbanos do Brasil está relacionado aos congestionamento e disponibilidade de vagas para estacionamentos, uma vez que não há um investimento notório no transporte de massa. Este trabalho apresenta o *Find* Parking Place*, um sistema ubíquo de gerência para estacionamentos, a fim de dar suporte ao usuário no sentido de escolher um estabelecimento para estacionar, baseado na disponibilidade de vagas, localização e preço.**

Index Terms*—Modelo de Arquitetura, Computação Móvel, Computação Distribuída, Computação Ubíqua, *Find* Parking Place

***Abstract*—The devices miniaturization with advent the wireless networks served as the basis for advent of research in ubiquitous and pervasive computing. In this context, various applications have been developed with focus on give support the several everyday life activities. A problem that has gained prominent in setting of large urban centers from Brazil is related to congestion and, availability of parking spaces, once there isn't investment notorious in mass transport. This work present the *Find* Parking Place*, a management ubiquitous system to parking, for give support the users in order to choose an establishement to park, based availability place, location and price.**

***Index Terms*—Architecture Model, Computing Distributed, Computing Ubiquitous, Find* Parking Place**

SUMÁRIO

I Introdução 1

II Computação Móvel 1

III Computação Distribuída 2

IV Modelos Arquiteturais 3
IV-A Modelo *Peer-to-Peer* 4
IV-B Modelo Cliente Servidor 4

V Computação Ubíqua e Pervasiva 4
V-A Computação Sensível ao Contexto e Serviços Baseados em Localização 5

VI Arquiteturas e Aplicações para Dispositivos Móveis e Localização de Estacionamentos - Estado da Arte 6
VI-A I - Análise de Arquiteturas 6
VI-B II- Análise de Aplicativos 7

VII Aspectos da computação ubíqua no projeto 8

VIII Arquitetura de Software 9

IX Modelagem de Dados 10

X Aplicação 11
X-A Plataforma Android 11
X-B Principais classes 11
X-C Funcionamento do Find* Parking Place 12

XI Web Service 14
XI-A Definição e Tecnologias utilizadas 14
XI-B Funcionamento do Web Service 16

XII Ambiente 17

XIII Resultados e Conclusão 17

XIV Agradecimentos 18

Referências 18

I. Introdução

Avanços no mundo tecnológico propiciaram o surgimento de computadores móveis com capacidade de se conectarem a Internet através de uma rede sem fio (*wireless*), ou seja, uma rede onde a comunicação entre estes computadores e a Internet é estabelecida através de ondas de rádio ou infravermelho. Isto possibilitou aos usuários acessarem recursos computacionais através de redes *wireless* heterogêneas, deste modo, um usuário pode continuar a acessar páginas da Web, redes sociais, fazer *downloads* e instalação de aplicativos, à medida que se desloca entre ambientes diferentes como sua casa e seu trabalho. A *mobile computing* (computação móvel) consiste na mudança de posição dos computadores proveniente da alternância de localização dos usuários, e sua difusão, serviu de alicerce para o crescimento das pesquisas em computação pervasiva e ubíqua.

Definida pela *International Business Machines Corporation* (IBM), em meados de 1990 [1], a computação pervasiva visa integrar os mundos computacional e real de modo transparente ao usuário. Esta integração é dada pela presença de pequenos dispositivos constituídos de hardware e software, com poder de processamento. Estes dispositivos podem estar inseridos em qualquer objeto do mundo real. Assim, objetos têm a capacidade de colher informações do ambiente real onde estão envolvidos e oferecer serviços computacionais para melhor atender as necessidades dos usuários [2].

Segundo [1], a computação ubíqua é considerada como a terceira era computacional, atrás da primeira era, dos *Main-Frames*, e da segunda era, dos *Personal Computers* (PCs). Esta computação também é considerada como a interseção dos paradigmas móvel e pervasivo, expondo assim, um ambiente que une a alta mobilidade do dispositivos ao alto grau de inteligência dos dispositivos embutidos no ambiente [2]. Neste ambiente, todos estes dispositivos e as demais tecnologias se "misturam a vida cotidiana das pessoas até se tornarem indistinguíveis" [3].

Um dos principais objetivos das aplicações ubíquas é auxiliar os usuários na prevenção e resolução de problemas cotidianos, como os relacionados aos congestionamentos e estacionamentos dos centros urbanos do Brasil. Tais problemas são ocasionados pelo aumento da quantidade de carros e a falta de infraestrutura nestas cidades, principalmente em datas comemorativas.

Visando facilitar a procura por vagas disponíveis nos estacionamentos, alguns centros e estabelecimentos comerciais estão investindo em estacionamentos inteligentes, ou seja, sistemas que identificam a disponibilidade das vagas através de sensores instalados no teto dos estacionamentos. Nesses sistemas, através de um painel eletrônico que fica localizado na entrada desses estacionamentos, os usuários ficam sabendo a quantidade das vagas disponíveis, bem como a localização das mesmas.

Aplicativos como *ParkLocator* [5], *BestParking* [6], *Onde-Parar* [7] e *Let'sPark* [8], foram desenvolvidos no intuito de auxiliar aos usuários a encontrar estacionamentos. Por meio desses aplicativos é possível identificar os endereços, preços por hora, situação, entre outras informações dos estacionamentos.

Contudo, os aplicativos mencionados apesar de informarem a localização dos estacionamentos, não apresentam o número de vagas disponíveis, e os usuários dos centros e estabelecimentos comerciais de algumas cidades são informados da localização e da quantidade das vagas disponíveis nos estacionamentos apenas no momento em que chegam aos mesmos.

O objetivo deste trabalho é desenvolver um sistema ubíquo de gerência para estacionamentos, baseado na disponibilidade de vagas, localização e preço dos mesmos, denominado de *Find* Parking Place*.

Neste sistema, o usuário poderá informar o bairro onde queira estacionar seu carro, e assim obter os estacionamentos localizados no mesmo. Os estacionamentos serão representados por ícones azuis ou vermelhos, indicando vagas disponíveis ou indisponíveis respectivamente. Ao clicar em um ícone o usuário será direcionado para uma tela do sistema que exibirá as informações do estacionamento escolhido.

O sistema proposto objetiva contribuir para o crescimento no desenvolvimento de sistemas móveis e distribuídos a fim de torná-los mais ubíquos para solucionar problemas do cotidianos dos usuários, tais como encontrar estacionamentos.

Este trabalho é organizado da seguinte forma: Na seção II é apresentada os principais aspectos da computação móvel. A seção III aborda conceitos da computação distribuída. Destaca-se na seção IV alguns dos modelos arquiteturais mais importantes para criação de sistemas distribuídos. Questões que envolvem os paradigmas Ubíquo e Pervasivo são abordadas na seção V. Na seção VI, são apresentados alguns trabalhos que apresentam características semelhantes ao trabalho proposto. Destaca-se na seção VII, pontos importantes da computação ubíqua envolvidos no projeto. Expõe-se na seção VIII a especificação da arquitetura definida para o trabalho proposto. Nas seções IX, X e XI, são evidenciados o modelos de dados, a aplicação *android* e o *web service*, respectivamente. Demonstra-se na seção XII, o ambiente do sistema proposto. Na seção XIII são apresentadas as conclusões do projeto e as sugestões para trabalhos futuros baseados na arquitetura deste trabalho.

II. Computação Móvel

Avanços tecnológicos propiciaram o surgimento da computação móvel, junto as redes sem fio, revolucionando a maneira de usar o computador [9]. Este paradigma consiste na mudança de posição dos computadores proveniente do deslocamento dos usuários, que continuam a acessar o serviço fornecido por estes computadores independente deste deslocamento [2].

Diferentemente da comunicação através da rede cabeada, a comunicação por *wireless* está mais propensa a desconexões, ou baixa largura de banda, devido a obstáculos e ruídos [9]. Isto acontece porque esta comunicação é estabelecida através de ondas de rádio ou infravermelho.

Um dos principais problemas da comunicação em redes sem fio é a segurança, pois obter uma conexão de acesso a esta

rede é mais simples do que em conexões por redes cabeadas, principalmente quando usuários têm autorização para acessar ambientes de domínio de segurança em grandes áreas.

Neste contexto, para prover um ambiente seguro em um meio de comunicação inseguro (rede sem fio), projetistas do *MIT* desenvolveram o *Kerberos* [1], cuja finalidade é autenticar usuários sem expor suas senhas na rede, bem como, autenticar os mesmos nos domínios onde eles são conhecidos.

A conexão via Internet é mais complexa para os computadores móveis, pois os computadores portáteis se conectam e desconectam a redes sem fio à proporção que se deslocam entre ambientes distintos.

A mudança de local dos computadores móveis enquanto estão conectados a Internet ocasiona no aumento de volatilidade da informação. Isso acontece pois o conjunto de componentes com os quais estes computadores se comunicam varia com esta mudança. Neste sentido, determinados dados estáticos para os computadores *desktop*, são considerados dinâmicos para os computadores móveis.

À alternância de endereço de rede de forma dinâmica devido a variação de localização dos dispositivos móveis não é suportada pela infraestrutura da Internet atual, a qual utiliza o Protocolo de Internet (IP), onde o nome do servidor está intrinsecamente ligado ao endereço de rede. Desta forma, esta alternância requer a obtenção de novos endereçamentos IP, sendo que, para gerir a mesma é necessário enviar mensagens para estes dispositivos a fim de obter suas posições atuais. Os quatro mecanismos necessários para obter tais posições são [9]:

- *Selectively broadcasting* - Consiste no envio de mensagem a todos os dispositivos na rede com o objetivo de obter como resposta o endereço atual dos mesmos. Entretanto, usar este mecanismo pode ser custoso se usado frenquentemente em uma grande rede, porém é praticável se o dispositivo móvel identificado está em um pequeno grupo de rede.
- *Central Services* - Neste mecanismo as posições atuais de cada computador móvel são mantidas em uma base de dados centralizada logicamente. A partir do momento em que estes computadores obtêm um nova posição envia-a através de mensagem para ser atualizada na base dados.
- *Home Base* - Mecanismo onde há uma migração do servidor central caso acesso a este esteja limitado ou inacessível pelos dispositivos.
- *Forwarding pointers* - Consiste em um conjunto de ponteiros na rede que encaminham, recebem as mensagens de mudança de determinado endereço do dispositivos e o atualizam na localização antiga.

O uso dos mecanismos mencionados permitem que os computadores móveis transportados por usuários utilizem endereços ou ponto de acesso de rede diferentes.

Atualmente os computadores móveis na forma de *smartphones*, *tablets*, entre outros, possuem cada vez mais capacidade de armazenamento, processamento, bem como modo econômico de energia das baterias, a fim de disponibilizar para os usuários dos mesmos mais acessibilidade aos serviços computacionais providos por estes dispositivos. Desta maneira, tais computadores estão sendo cada vez mais difundidos, onde o Brasil tem cerca de 70 milhões de *smartphones*, ficando atrás apenas da China, Estados Unidos e Japão [10].

[1]Disponível em: http://web.mit.edu/kerberos/

III. Computação Distribuída

A computação distribuída é constituída de vários computadores independentes que compartilham recursos através de mensagens, por meio de uma rede de computadores [11]. Tais computadores fornecem serviços aos usuários como se estes fossem oriundos de apenas um computador, e por estarem os mesmos interligados pela rede, podem trocar mensagens estando separados entre quaisquer distâncias, ocasionando os seguintes aspectos como consequência:

- *Concorrência* - Estabelece o modo dos computadores compartilharem recursos nas redes de computadores paralelamente. Por exemplo, é possível que duas pessoas trabalhando em locais separados possam compartilhar arquivos, paginas web, dados obtidos de uma base de dados, entre outros.
- *Inexistência de Relógio Global* - Consiste na inexistência de precisão global única do tempo correto, em que os computadores podem sincronizar seus relógios em uma rede[11].
- *Falhas Independentes* - Consiste na falha de um determinado computador na rede sem ser percebido imediatamente pelos os outros com qual o mesmo se comunica.

Define-se também como computação distribuída o modo como os processos interagem através de troca de mensagens em um mesmo *host* (computador), a fim de executar as tarefas do sistema.

A construção de sistemas distribuídos é motivada pelo desejo de compartilhar recursos através da rede, onde tais recursos abrangem uma série de objetos que podem ser compartilhados pela rede, tais como, páginas da web, arquivos de áudio e vídeo, informações de um determinada base de dados, entre outros. Diante disto, são descritos a seguir alguns exemplos de sistemas distribuídos.

Um exemplo de um amplo sistema distribuído é a Internet, utilizada por cerca de 2,7 bilhões de pessoas no mundo [12]. A Internet é composta por diferentes redes interligadas as quais possuem vários computadores conectados.

Sistemas construídos para dispositivos móveis cuja finalidade é o compartilhamento de recursos através da rede sem fio, também são definidos como sistemas distribuídos.

Os sistemas ubíquos também podem ser definidos como distribuídos. Nesses sistemas, dispositivos com alto grau de mobilidade interligam-se a computadores com um elevado grau de presença e percepção do ambiente em que estão envolvidos, compartilhando recursos por meio da rede, se tornando invisíveis aos usuários desse ambiente.

Observa-se que o número de sistemas distribuídos construídos para serem executados sobre a Internet aumenta diariamente principalmente depois do surgimento das tecnologias

móveis, pervasivas e ubíquas, entretanto para contruí-los é preciso levar em consideração aspectos como escalabilidade, tratamento de falhas, concorrência e transparência.

Define-se um sistema distribuído como escalável quando o mesmo continua sendo eficaz à medida que o número de usuários e recursos aumenta, apresentando os seguintes desafios em sua construção:

- *Controlar o custo dos recursos físicos* - Consiste na ampliação do sistema para atender um aumento da demanda por recurso. Neste sentido, para um sistema com n usuários exige-se o máximo de O(n) de recursos para que o sistema continue escalável [11]. Sendo assim, se um servidor de arquivos suporta o acesso de 15 usuários, dois servidores, para manter a escalabilidade têm que suportar o acesso de 30 usuários.
- *Controlar perda de desempenho* - Considerando o aumento no conjunto de dados presentes na tabela que contém os nomes de domínio e seus endereços IP correspondente, mantida pelo DNS (*Domain Name System*), que possui algoritmos de busca de estruturas hierárquicas para resolver tais endereços em nomes. A perda de desempenho para que o sistema permaneça escalável diante deste crescimento, não pode ser maior que O($log\ n$) [11].
- *Impedir que recursos de software se esgotem* - Um exemplo de recurso que está se esgotando, e portanto um falta de escalabilidade, são os números que utilizam 32 bits nos endereçamentos IP, denominados de IPv4. Para contornar está situação estão sendo adotadas novas faixas de endereço com 128 bits também conhecido como IPv6, o que exigirá a mudança de muitos componentes de software [11].
- *Evitar gargalo de desempenho* - Define a descentralização de um serviço, por exemplo, um servidor central que passa a ser acessado por muitos usuários, pode ter sua a informação compartilhada com outros servidores na rede o para diminuir o gargalo.

Como mencionado anteriormente, um sistema distribuído pode continuar funcionando mesmo que os componentes falhem na rede. Deste modo tratar falhas de tais componentes é particularmente difícil [13]. Mecanismos para o tratamento de falhas em sistemas distribuídos são expostos a seguir.

- *Detecção de falhas* - Nem todas as falhas podem ser detectadas, mecanismos como somas de verificação podem ser usados para identificação de dados que estão presentes em mensagens ou arquivos de forma danificada.
- *Tolerância a falhas* - Consiste em serviços utilizados para contornar o aparecimento de falhas. Tais serviços serão apresentados no item Redundância, sequencialmente.
- *Recuperação de falhas* - Compreende projetar o sistema para que o "estado dos dados permanentes possa ser recuperado ou retrocedido" [13].
- *Redundância* - Define o modo como os serviços usam componentes redundantes para tolerar falhas. Exemplos destes componentes são explicados nos subitens a seguir.
  1) Sejam dois roteadores quaisquer na Internet os mesmos devem está sempre interligados por duas rotas diferentes. Neste sentido, caso uma rota falhe a outra poderá ser utilizada.
  2) Toda tabela de correspondência de nomes e endereços IP do *Domain Name System* tem que ser replicada em dois servidores distintos.
  3) Para contornar falhas de acesso aos servidores de banco dados pode-se replicá-los em vários servidores. Desta forma, caso ocorra falha em um dos servidores, os clientes são redirecionados para os outros, garantindo o acesso aos dados do banco.

Serviços e aplicativos podem fornecer recursos para serem compartilhados em um sistema distribuído, onde há sempre a possibilidade de que vários clientes queiram acessar um mesmo recurso compartilhado ao mesmo tempo, ocasionando um problema de concorrência.

Define-se como transparência, o modo com o qual é abstraído do usuário final ou do desenvolvedor de aplicativos, a separação dos componentes de um sistema distribuído, dando a visão ao mesmo do sistema como um todo. As principais transparências são as de acesso e localização, pois a presença ou ausência destas transparências "afetam fortemente a utilização dos recursos distribuídos" [11], as quais são descritas a seguir.

- *Transparência de acesso* - Consiste na permissão de acesso a recursos remotos e locais de forma idêntica. Por exemplo, no sistema para armazenamento de arquivos em nuvem *DropBox*[2], é possível que o usuário veja a mesma estrutura de pastas e diretórias tanto local como remotamente.
- *Transparência de localização* - Ocorre quando os recursos são acessados sem saber da localização física de onde se está acessando. Um exemplo disto pode ser ilustrado pela URL: http://www.wiki.ifba.edu.br/gsort/tiki-index.php; onde a parte da URL(www.wiki.ifba.edu.br), que identifica o nome do servidor web, se refere ao nome de um computador em um domínio, e não ao endereço IP.

IV. Modelos Arquiteturais

Em sistemas distribuídos os modelos arquiteturais expõem o posicionamento dos componentes do sistema e o modo como eles se relacionam através da rede de computadores. Tais modelos são compostos por camadas de serviços, que definem a forma de como os serviços são fornecidos e solicitados por processos contidos em mesmo computador ou em computadores diferentes [11]. A Figura 1 ilustra as camadas de serviço em sistemas distribuídos, as quais são descritas a seguir, e posteriormente serão destacados os principais modelos de arquitetura destes sistemas.

- *Aplicativos, serviços* - Consiste nos serviços oferecidos por meio da interação de um ou mais processos servidores, bem como da interação com os processos clientes,

[2]https://www.dropbox.com

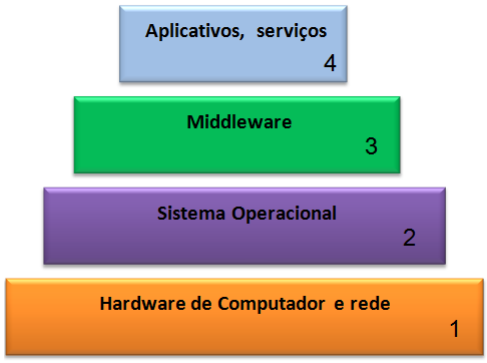

Figura 1. Camadas de serviço de sistemas distribuídos (adaptado de [11])

no intuito de manter um nível estável dos recursos destes serviços. Um exemplo disto é o protocolo NTP (*Network Time Protocol*) implementado na Internet, utilizado para realizar o mecanismo de sincronização de relógios de computadores por meio de processos servidores que são executados em diversos computadores na Internet, os quais por meio de interação com outros servidores acertam a hora solicitada por processos clientes [11].

- *Plataformas* - Compreende as camadas 1 e 2 da figura 1 que são as camadas de hardware e software de baixo nível que têm a função de fornecer serviços para as camadas de mais alto nível. São exemplos de plataformas, Intel x86/Windows , Intel x86/Linux, entre outras [11].
- *Middleware* - Camada de software, cujo objetivo é ocultar as diferenças das plataformas de baixo nível, fornecendo um modelo comum para programação de aplicações distintas. RPC (*Remote Prodedure Call*) e Isis foram os primeiros tipos de *middleware* implementados, e estão sendo utilizados os orientados a objetos como CORBA, RMI Java, Serviços Web, entre outros. No modelo deste trabalho foi adotado o *web service* como *middleware* [11].

A. Modelo Peer-to-Peer

Caracteriza-se o *peer-to-peer* como um modelo onde não há diferenciação entre processos clientes e servidores, bem como não existe distinção em que computadores tais processos são executados, os quais interagem com outros processos como pares. Ilustra-se na Figura 2 um modelo de um aplicativo *peer-to-peer* que possui vários processos (pares) executados em diferentes computadores, que por sua vez, compartilham recurso (objetos) entre si visando prover serviços aos usuários [11].

Um exemplo de implementação deste modelo é o sistema *uTorrent* [3], que permite os usuários baixar arquivos em seus computadores, que por sua vez, serão enviados para os computadores de outros usuários que requisitarem estes mesmos arquivos.

[3]http://www.utorrent.com/intl/pt/

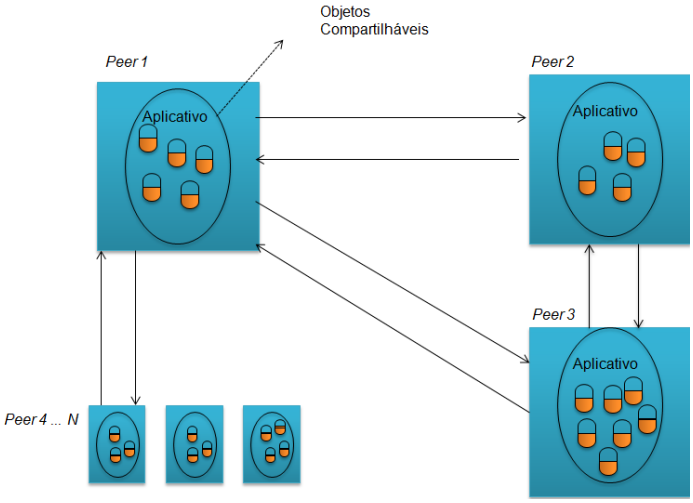

Figura 2. Modelo de Arquitetura *Peer-to-Peer*

B. Modelo Cliente Servidor

Modelo de arquitetura cliente servidor considerado o mais importante em relação aos sistemas distribuídos 3, onde os processos clientes interagem com os servidores por meio de mensagens no intuito de obter os recursos compartilhados dos mesmos [11].

Observa-se neste modelo que os servidores podem se tornar clientes de outros servidores. Por exemplo, um cliente que requisita informações de arquivos a um servidor de aplicação, que por sua vez requisita tais informações a um servidor de arquivos. A Arquitetura desenvolvida neste projeto (VIII) baseia-se neste modelo.

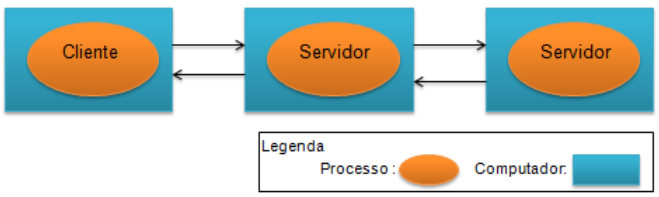

Figura 3. Modelo de Arquitetura Cliente Servidor

V. Computação Ubíqua e Pervasiva

A forma com a qual as pessoas convivem com os computadores pode ser dividida em três eras. A primeira, era dos grandes mainframes que ocupavam grandes espaços constituídos de válvulas e relés, onde este único computador servia a várias pessoas. A segunda era inicia-se com surgimento dos PCs (*Personal Computers*), a qual cada usuário interagia e acessava o serviço computacional do seu próprio computador estacionário. A terceira era é definida nos dias atuais, onde cada usuário possui e está rodeado de vários computadores interligados através de redes sem fio no ambiente onde está situado, podendo interagir com os mesmos de forma invisível em seu cotidiano. Denomina-se esta interação entre usuários

computadores e o ambiente no qual estão inseridos de Computação Ubíqua [1].

O termo computação ubíqua foi definido por *Mark Weiser* em 1988 nos laboratórios da Xerox *Park*, o qual vislumbrava um mundo onde as pessoas viveriam cotidianamente com as tecnologias computacionais, ou seja, nas atividades domésticas, no trabalho, no lazer entre outros. Para *Weiser* tais tecnologias iriam se "misturar a vida cotidiana das pessoas até se tornarem indistinguíveis"[3].

Neste ambiente repleto de tecnologias, *Weiser* percebeu a oportunidade dos usuários mudarem o estilo com o qual acessavam o serviço computacional adquirido desde a era dos mainframes e transportado para era dos PCs, onde os mesmos ficavam em frente ao computador, olhando para tela com as mãos no teclado e mouse. Neste estilo de interação os usuários perdem a noção do que está acontecendo ao seu redor ficando apenas concentradas na atividade que está sendo desenvolvida em frente ao computador.

Sendo assim, diferentemente deste estilo de interação, no mundo ubíquo vislumbrado por *Weiser* as pessoas poderiam associar suas atividades cotidianas as desempenhadas no computador, de tal forma que esta interação se tonaria invisível.

Visando o entendimento deste termo definido pelo precursor da *Ubicomp* (Computação Ubíqua), considera-se uma analogia com uma simples caneta esferográfica. Esta caneta é constituída basicamente de um tubo com uma esfera de tungstênio (um tipo de material), de meio milímetro na ponta, que ao ser inserida sobre o papel, gira sugando a tinta do tubo e depois passa esta tinta no papel gerando a escrita [14].

As canetas esferográficas são utilizadas por milhares de pessoas no mundo todo cotidianamente, entretanto elas se preocupam apenas em usar tais canetas para simplesmente escrever, fincando assim invisível a tecnologia usada para gerar a escrita no papel. Por outro lado, os computadores estacionários tradicionais raramente fazem isto, eles obrigam o usuário a ter sempre que configurá-los de alguma forma para concluírem uma determinada tarefa.

Um outro termo usado para definir a *Ubicomp*, também foi a "*calm technology*"[1][15]. Um exemplo desta tecnologia seriam uso dos emergentes óculos inteligentes, onde o individuo consegue ver e captar informações de onde está inserido de uma maneira mais leve do que se fosse através de um PC, onde o mesmo iria procurar vários sistemas e diversos sites para obter as mesmas informações.

Estudos no ramo da Computação Ubíqua contribuíram para o surgimento da Computação Sensível ao contexto [3], que tem como objetivo fornecer serviços às pessoas a partir de uma coleta de informações chamadas de contexto de uso. Sendo que, esta coleta é feita no cenário em que estas pessoas estão envolvidas, por dispositivos como, câmeras celulares, sensores, redes sem fio (*Bluetooth* e *Wi-Fi*), etc. Para exemplificar esta computação consideram-se os estacionamentos inteligentes citados na seção introdutória deste trabalho, os quais disponibilizam as vagas disponíveis aos seus usuários por meio da coleta de informações através de sensores ultrassônicos inseridos em cada vaga, que identificam quando um automóvel as ocupa.

Alguns estudiosos da computação definem Computação Ubíqua e Computação Pervasiva como sinônimos, entretanto estes dois paradigmas são distintos conceitualmente [2]. Denominado pela IBM em meados de 1990, a Computação pervasiva visa expor um ambiente envolvido de computadores embarcados com a capacidade de obter informações deste ambiente onde estão inseridos, construindo modelos computacionais inteligentes para usuários de forma invisível, auxiliando-os no dia a dia.

A IBM foi uma das primeiras empresas a investir em implementações comercias voltadas para Computação Pervasiva, tais como a implementação criada em 1999 em parceria com a empresa *Swissair*, que possibilitava o passageiro fazer *check-in* usando seu celular que por sua vez, servia também de cartão de embarque, apresentando informações como assento, número e horário do voo entre outros [1]. Também denominada de interseção da computação Móvel com a Computação Pervasiva, a *Ubicomp* une estes dois paradigmas expondo um ambiente com alta mobilidade e um alto grau de dispositivos inseridos no mesmo, construindo assim, modelos computacionais inteligentes em todos lugares de forma auxiliar os usuários cotidianamente de foma indistinguível. Neste sentido, pode-se afirmar que a Computação Ubíqua é Móvel e Pervasiva, entretanto a recíproca de ambas para com a *Ubicomp* não é sempre verdadeira.

A. Computação Sensível ao Contexto e Serviços Baseados em Localização

Estudos no ramo da computação ubíqua contribuíram para o surgimento da computação sensível ao contexto [3], que tem como objetivo fornecer serviços aos usuários a partir de uma coleta de informações chamadas de contexto de uso. Pesquisas na área da computação sensível ao contexto, juntamente com as pesquisas na área da computação móvel e o surgimento das tecnologias emergentes anteriormente mencionadas, propiciaram o crescimento das aplicações de Serviços Baseados em Localização (SBL).

Os SBL têm como objetivo auxiliar os usuários provendo informações geográficas tanto através dos dispositivos móveis carregados por eles, como por meio dos dispositivos dispostos no ambiente onde os mesmos estão localizados [16].

Existem várias formas de classificar os SBL, uma das mais importantes é o modo de como o usuário recebe a informação, que pode ser *push* ou *pull* [17]. No modo *push* o usuário recebe a informação de acordo com sua localização. Por exemplo, o usuário poderia receber a informação do transito. No modo *pull* o usuário precisa requisitar a informação. Em uma aplicação deste modo, um usuário poderia requisitar a localização de um determinado cinema [17].

Neste contexto nota-se que o ambiente ubíquo tem a capacidade de integrar Computação Móvel, Computação Pervasiva, Computação Sensível ao Contexto, Serviços Baseados em Localização, entre outros, por meio de diversas redes heterogêneas, tais como as redes sem fio de longa, média e curta distância. Isto significa que este ambiente pode interligar diversos e diferentes softwares, tecnologias e plataformas

auxiliando e interagindo com os usuários diariamente de forma imperceptível a eles.

Neste sentido, as aplicações como *waze* considerado como um dos melhores aplicativos de trânsito do mundo, o qual informa o congestionamento do trânsito nas ruas, avenidas e estradas dos grandes centros urbanos, bem como indica as rotas de trânsito livre para os seus usuários, está intrinsecamente envolvido em um ambiente ubíquo [18].

VI. Arquiteturas e Aplicações para Dispositivos Móveis e Localização de Estacionamentos - Estado da Arte

Para construir a arquitetura do sistema proposto foram utilizados dois critérios de análise: (I) análise das arquiteturas; (II) análise de aplicativos.

A. I - Análise de Arquiteturas

A seguir são descritas as arquiteturas analisadas.

1) Arquitetura Genérica para Dispositivos Móveis: Apresenta-se na Figura 4, uma arquitetura genérica de software para disponibilização de uma aplicação web para dispositivos móveis, cuja a finalidade é possibilitar que uma aplicação web em execução no servidor possa ser disponibilizada para também ser executada em dispositivos móveis clientes, independente de quais sejam tais dispositivos, dos sistema operacionais e servidores dos mesmos, e também do protocolo de comunicação de rede sem fio utilizado [4]. Esta arquitetura foi dividida em dois ambientes, Fixo e Móvel, os quais serão explicados a seguir.

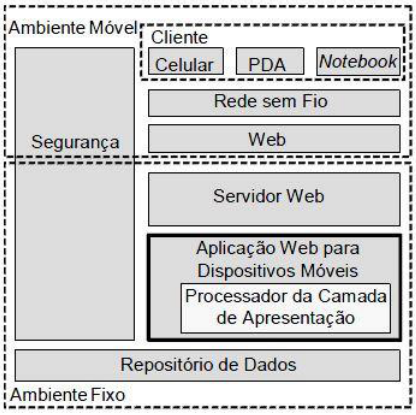

Figura 4. Modelo de Arquitetura para Dispositivos Móveis [4].

O ambiente Fixo é composto dos componentes:

- *Aplicação Web para Dispositivos Móveis* - Através deste componente as páginas web são disponibilizadas em um servidor, as quais são acessadas por computadores *desktop* e dispositivos móveis, que necessitam de uma adaptação de conteúdo por terem características menores em relação aos computadores *desktop*, tais como, telas, teclados, memória, bem como, poder de processamento limitados.
- *Processador da Camada de Apresentação* - Componente principal da arquitetura que tem a função de separar o conteúdo a ser apresentado na página web do formato de apresentação que pode ser nas linguagens (WML, HTML e XHTML), de acordo com a linguagem que o cliente requisitar.

O ambiente móvel é constituído pelos variados dispositivos móveis do mercado, tais como: *PDAs*; *Smartphones*; *Tablets*; *Notebooks*; entre outros; sendo que a é necessário que os mesmos tenham um serviço para navegação na Web.

O modelo arquitetural proposto neste trabalho possui características semelhantes a arquitetura para dispositivos móveis (Figura 4) tais como, a utilização do modelo cliente-servidor, foco no lado do cliente por meio de dispositivos móveis, interação com ambiente externos entre outros. Essa arquitetura se mostra mais independente da arquitetura deste projeto, pois tem todo o processamento do lado do servidor, ficando a cargo dos distintos clientes apenas consumir. Entretanto, a arquitetura deste projeto é pouco independente em relação a esse outro modelo, pois esta voltada atualmente apenas para dispositivos que utilizam o Sistema Operacional *Android*, porém, ganha na construção de aplicações mais ricas através dos recursos oferecidos por este sistema.

2) Modelo de Arquitetura do ParkLocator: Através do modelo arquitetural do *ParkLocator* (Figura 5), o usuário pode interagir com o ambiente onde se encontra por meio do seu *smartphnoe*, e obter informações dos estacionamentos de acordo com sua posição, baseadas em: preço; hora limite de permanência na vaga; localização do usuário; a partir de alarmes, notificações e mensagens sonoras. Em sequencia serão apresentadas os principais componentes desta arquitetura.

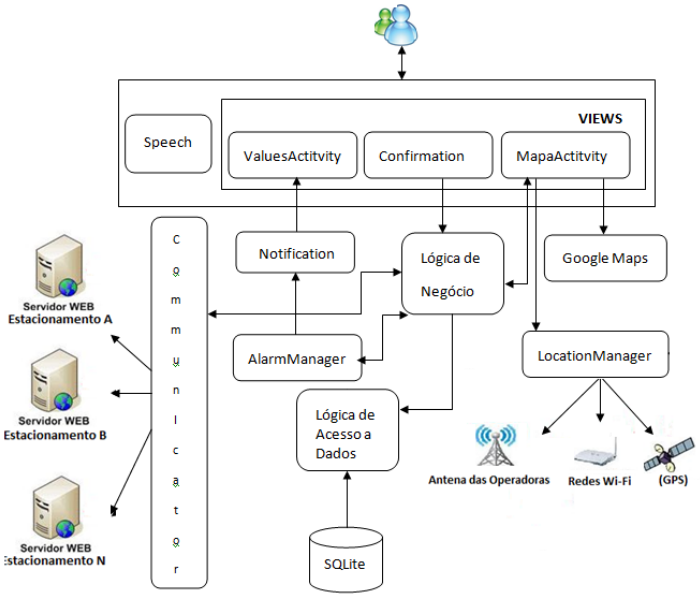

Figura 5. Modelo Arquitetural do ParkLocator [5].

- *Speech* - Componente responsável por converter texto em um som. Na aplicação *ParkLocator* o *Speech* é utilizado para enviar uma notificação sonora ao usuário no momento em que um estacionamento é encontrado.

- *AlarmManager* - Componente que tem a função de agendar um alarme para lembrar ao usuário, por exemplo, que a hora de utilização do estacionamento falta 10 minutos para vencer.
- *Notification* - Componente responsável por enviar uma mensagem de notificação ao usuário sempre que a utilização do estacionamento estiver prestes a vencer.
- *Communicator* - Responsável pela comunicação entre os servidores web e a aplicação *Android*, onde este ultimo requisita ao primeiro os dados do estacionamento encontrado via JSON (*JavaScript Object Notation*).

Este modelo de arquitetura serviu de referência para o modelo proposto, pois além de ser voltado para a localização de estacionamentos em centros urbanos, faz uso de SBL, interação com ambientes externos através de *web service* e tem como base a plataforma *android.*

3) Tabela de Características Relevantes das Arquiteturas: A Tabela I ilustra as principais características relevantes entre as arquiteturas estudadas. A ultima coluna foi colocada em destaque para uma melhor identificação da arquitetura deste projeto.

Tabela I
COMPARATIVO DE CARACTERÍSTICAS RELEVANTES DOS MODELOS ESTUDADOS

Comparativo	**Arquitetura para Dispositivos Móveis**	**Arquitetura ParkLocator**	**Arquitetura Find* Parking Plcae**
Modelo de Arquitetura	Cliente-Servidor	Cliente-Servidor	**Cliente-Servidor**
Voltada para	Desktops e Dispositivos Móveis	Dispositivos Móveis	**Dispositivos Móveis**
Uso de SBL	Não	Sim	**Sim**
Interação com sistemas externos	Sim	Sim	**Sim**
Área de Processamento	Servidor	Dispositivo Móvel	**Web Service e Dispositivo Móvel**
Notificação do usuário	Não	Sim	**Sim**
Alertas Sonoros	Não	Sim	**Não**
Indicativo de direcionamento	Não	Não	**Sim**

B. II- Análise de Aplicativos

A seguir são apresentados os aplicativos analisados.

1) BestParking: Presente em cerca 105 cidades e 115 aeroportos na América do Norte a (*app* do inglês *application*) *BestParking*, auxilia os usuários a encontrar os estacionamentos mais convenientes para os mesmos. Através desta *app* os usuários ficam sabendo dos valores, localização, fotos, entre outros, bem como as rotas que serão percorridas a partir das suas posições até aos estacionamentos. A busca para obter tais informações pode ser feita pela localização atual dos usuários,

Figura 6. BestParking

endereço, cidade, bairro, bem como, por aeroportos dentre outros. O *BestParking* fornece os dados de alguns estacionamentos gratuitamente, neste sentido, para visualizar mais estacionamentos os usuários precisam desembolsar US$2,00. O Aplicativo pode ser baixado para as plataformas *Android* [6] e *iOS* [19].

O *BestParking* possui características semelhantes ao *Find* Parking Place* desenvolvido neste projeto: indicação de direcionamento, valor hora, utilização de GPS para buscar a localização dos usuários e busca por endereço. O *BestParking* fornece uma maior variedade de busca que o *Find* Parking Place*, que pesquisa apenas pela localização atual e bairro do usuário. Entretanto o *Find* Parking Place* por ser mais compacto necessita de menos auxilio do usuário para obter os estacionamentos.

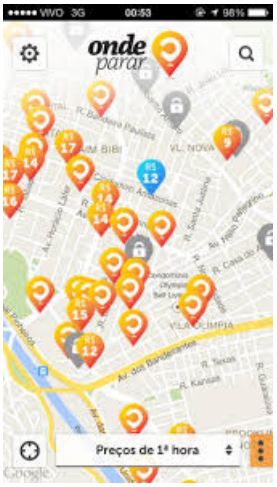

Figura 7. OndeParar

2) OndeParar: OndeParar é um aplicativo gratuito que informa ao usuário quais estacionamentos estão mais próximos do mesmo, bem como, preços, forma de pagamento, entre outros. Este aplicativo também informa as rotas e a distância entre o usuário o estacionamento pretendido pelo mesmo, e

através do cadastro de perfil no aplicativo este usuário poderá incluir informações, como avaliação de lugares e atualização de preços. Atualmente este aplicativo pode ser encontrado nas plataformas *Android* [7] e *iOS* [7] [20].

Assim como o *BestParking*, o *OndeParar* também tem particularidades similares ao *Find* Parking Place*, tais como: (i)mostrar os estacionamentos próximos ao usuário no momento que o mesmo inicia a aplicação; (ii) apresentar os preços; (iii) as rotas entre o estacionamento e a posição atual do usuário. Este aplicativo permite que usuários incluam informações sobre os estacionamentos, agregando mais informações para a base de dados do sistema. O *Find* Parking Place* não possui este mecanismo, no entanto, pode evitar que o usuários insiram dados inconsistentes na base de dados do sistema.

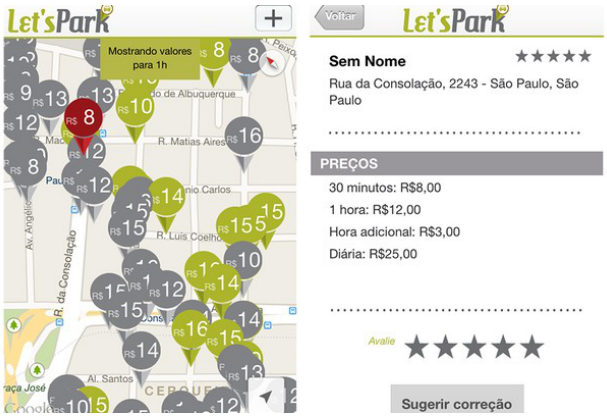

Figura 8. Let'sPark

3) Let'sPark: Diferentemente das aplicações mencionadas, o *Let'sPark* é um aplicativo gratuito que não oferece o mecanismo de busca por endereço. A *app* exibe através do *google maps* os estacionamentos contendo os endereços e preços por hora de acordo com a posição dos usuários. No mapa são exibidos marcadores da cor verde, vermelha e cinza que indicam respectivamente se os estacionamentos estão abertos, fechados ou se não existem informações detalhadas.

O aplicativo também funciona como uma ferramenta colaborativa, onde os usuários podem inserir informações sobre novos estacionamentos na cidade. Na *app* é possível obter informações sobre os estacionamentos da cidade de Salvador e pode ser encontrada tanto plataforma *Android* [8] como na *iOS* [21].

De forma similar ao *Find* Parking Place* este aplicativo também exibe ícones para identificação dos estacionamentos no mapa, porém na aplicação desenvolvida neste trabalho, os ícones verdes indicam os estacionamentos que contem vagas e os em vermelho que não tem vagas.

4) ParkLocator: Baseado na arquitetura supramencionada, o *ParkLocator* (Figura 9) é uma aplicação sensível ao contexto também voltada para localizar estacionamentos em centros urbanos. Esta aplicação informa aos usuários através de notificações, ícone, valores por hora cobrado, nomes, entre outros, dados dos estacionamentos mais próximos dos mesmos.

Figura 9. ParkLocator

Dentre estes dados os nomes dos estacionamentos encontrados são informados a estes usuários através de mensagem de voz quando este está a cerca de 10 metros destes estacionamentos, os quais após estacionarem seus carros, também recebem um alerta 10 minutos antes da hora de utilização dos estacionamentos vencerem.

A aplicação *ParkLocator* também apresenta características semelhantes ao *Find* Parking Place*, as quais podem ser destacadas: a exibição dos estacionamentos próximos aos usuários; os ícones representado os estacionamentos no mapa; os preços dos estacionamentos. Diferentemente da aplicação desenvolvida neste trabalho, o *ParkLocator*, informa os nomes dos estacionamentos através mensagens de voz. Por outro lado, o *Find* Parking Place* possibilita o usuário traçar rota entre a posição do usuário e o estacionamento pretendido pelo mesmo.

5) Tabela de Características Importantes das Aplicações: A Tabela II ilustra um comparativo entre as características importantes das aplicações supracitadas com a aplicação *Find* Parking Place* exposta detalhadamente na Seção X, desenvolvida baseada na arquitetura apresentada na Seção VIII. A última coluna em destaque identifica a aplicação desenvolvida neste trabalho.

VII. Aspectos da Computação Ubíqua no Projeto

A seguir serão apresentados tópicos a fim de esclarecer as características da computação ubíqua, bem como o uso das áreas do conhecimento envolvidas na mesma, tendo como base este projeto.

- *Invisibilidade* - Ao iniciar a aplicação *Find* Parking Place*, o usuário pode receber informações dos estacionamentos do bairro onde está localizado, sem que o mesmo precise informar algum parâmetro para a aplicação. As tecnologias e sistemas integrados ao *Find* Parking Place*, tais como, *web service*, e o SGBD, são transparentes para o usuário.
- *Heterogeneidade* - O sistema proposto interliga tecnologias e plataformas distintas, dentre as quais pode-se

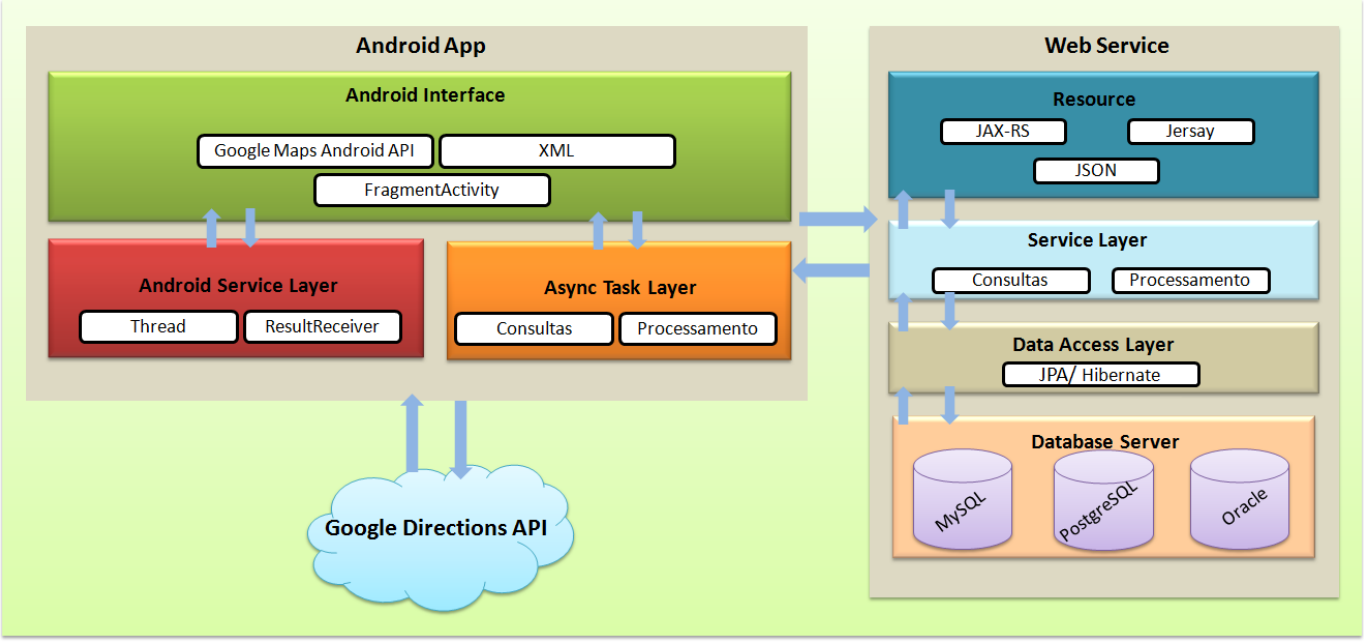

Figura 10. Arquitetura de Nível 1

Tabela II
COMPARATIVO DAS CARACTERÍSTICAS RELEVANTES DAS APLICAÇÕES ESTUDADAS

Comparativo	**Best-Parking**	**Onde-Parar**	**Lets'-Park**	**Park-Locator**	**Find* Parking Place**
Plataforma de Desenvolvimento	Android e iOS	Android e iOS	Android e iOS	Android	**Android**
Uso de ícones	Sim	Sim	Sim	Sim	**Sim**
Uso de GPS	Sim	Sim	Sim	Sim	**Sim**
Permite Traçar Rota?	Sim	Sim	Não	Não	**Sim**
Funciona como Ferramenta colaborativa?	Não	Sim	Sim	Não	**Não**
Atualiza Quantidade de Vagas	Não	Não	Não	Não	**Sim**

destacar, dispositivos móveis, base de dados, *web service*, servidor de aplicação, entre outros.

- *Serviços Baseado em Localização* - Todo o serviço da aplicação, envolvendo mapa, mecanismo de rota, entre outros, foi utilizado o GPS do *smartphone* do usuário.

VIII. ARQUITETURA DE SOFTWARE

Para desenvolver o sistema com características de mobilidade, distribuição e ubiquidade foi necessário a criação de um modelo de arquitetura em três camadas cliente-servidor. A Figura 10 ilustra esta arquitetura, onde a primeira camada é representada pela aplicação *Android*, a segunda pelo servidor de aplicação onde é executado o *web service* e a terceira pelo servidor de banco de dados. Os componentes desta arquitetura são descritos a seguir.

- *Android interface* - Consiste em um conjunto de bibliotecas e tecnologias responsáveis por gerar e gerenciar a interface desta aplicação, a qual exibe mapas, botões de busca, entre outros, por conta da integração de tecnologias e bibliotecas, tais como *Extensible Markup Language* (XML), *Google Android Maps API* e *Activitys*, explicadas mais detalhadamente na Seção X.
- *Android Service Layer* - Componente responsável por executar processos em segundo plano, também explicado na Seção X.
- *Async Task Layer* - Define um conjunto de tarefas na aplicação que são executadas de forma assíncrona, a fim de não comprometer a execução da tarefa principal.
- *Google Directions API* - Serviço que determina o calculo de rotas entre endereços, no caso da aplicação, entre o endereço do usuário do aplicativo e o do estacionamento requisitado pelo estacionamento.
- *Resource* - Componente presente no *web service* que através das tecnologias *JAX-RS* e *Jersey* (ver seção XI), converte os dados dos estacionamentos para *JavaScript Object Notation JSON* antes de enviá-los aos clientes, ou seja, usuários utilizando a aplicação através de seus *smartphones*.
- *Service Layer* - Camada intermediária entre as camadas *Resourse* e *Data Access Object DAO*, responsável pelo recebimento, processamento e envio das informações referentes aos estacionamentos.
- *Data Access Layer* - Camada de acesso a dados que uti-

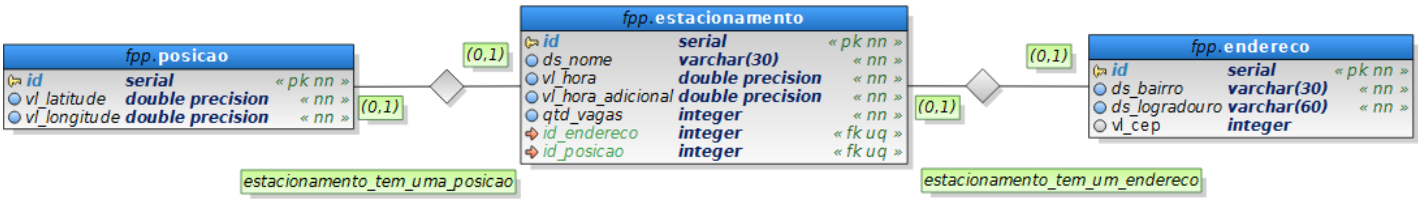

Figura 11. Modelagem de Dados

liza a especificação *Java Persistence API* (*JPA*) junto com a ferramenta de mapeamento objeto relacional (ORM) *Hibernate*, para mapear os objetos a serem persistidos no banco de dados do projeto.
- *Database Server* - Define os servidores de banco de dados do projeto onde serão obtidas as informações dos estacionamentos.

IX. Modelagem de Dados

Observa-se na Figura 11, que o modelo foi criado sob um *schema* denominado de *fpp* por causa do nome do sistema (*Find* Parking Place*), cujo o objetivo é agrupar as entidades do modelo, com seus respectivos relacionamentos. Tais entidades representam os objetos do mundo real, porém abstraindo a complexidade dos mesmos, haja vista que estas entidades não possuem a especificidade deses objetos reais. Estas tabelas, seus atributos e relacionamentos serão explicados detalhadamente a seguir.

Expõe-se na Figura 12, a tabela *posicao*, responsável por conter as informações inerentes a posição absoluta de um determinado estacionamento. Os atributos desta tabela serão descritos da seguinte forma:

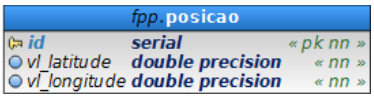

Figura 12. Tabela Posicao

- *id* - Consiste em um atributo numérico auto sequencial, que é utilizado como chave primária na tabela;
- *vl_latitude* - Atributo decimal cujo objetivo é armazenar os dados referentes a latitude;
- *vl_longitude* - Campo decimal onde será mantido os dados relacionados a longitude.

Exibe-se na Figura 3, a tabela *endereco*, cuja a função é manter os dados pertencentes ao endereço de um determinado estacionamento. Os atributos desta tabela são apresentados sequencialmente.

Figura 13. Tabela Endereco

- *id* - Assim como a entidade *posicao*, esta tabela também possui um atributo numérico auto sequencial, o qual é utilizado como chave primária;
- *ds_bairro* - Campo caractere responsável por armazenar a descrição do bairro;
- *ds_logradouro* - Atributo caractere onde será mantido os dados relativos a descrição do logradouro;
- *vl_cep* - Define o valor numérico que conterá as informações do Código de Endereçamento Postal (CEP).

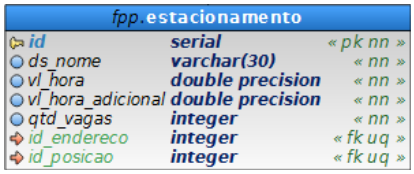

Figura 14. Tabela Estacionamento

Apresenta-se na Figura 14, a tabela *estacionamento* que é a principal tabela do modelo, onde serão mantidas as informações referentes a um determinado estacionamento e que é constituída dos seguintes atributos:

- *id* - Assim como nas tabelas citadas nas Figuras 12 e 13, esta tabela também é composta de um atributo numérico auto sequencial, que é utilizado como chave primária;
- *ds_nome* - Campo do tipo carácter onde será mantido o nome do estacionamento;
- *vl_hora* - Atributo decimal responsável por armazenar o valor relativo as horas e minutos gastos ao estacionar um veículo;
- *vl_hora_adicional* - Campo decimal cujo a função é armazenar um valor adicional ao *vl_hora*, caso estas horas e minutos definidas neste campo, tenham sido extrapoladas;
- *qtd_vagas* - Define a quantidade de vagas do estacionamento;
- *id_endereco* - Consiste num atributo numérico que representa a chave estrangeira para a tabela *endereco*;
- *id_posicao* - Assim como o campo referenciado anteriormente *id_endereco*, este campo também é uma chave estrangeira, porém faz referência a tabela *posicao*.

Ilustra-se também na Figura 11, o relacionamento entre as entidades *Estacionamento* e *Posicao* denominado de *estacionamento_tem_uma_posicao*, informando que todo estacionamento terá uma posição absoluta. Este relacionamento é definido pela chave estrangeira *id_posicao*, definida anteriormente.

Assim como o relação mencionada, demonstra-se na Figura 11, o relacionamento entre as entidades *estacionamento* e *endereco* nomeado de *estacionamento_tem_um_endereco*, informando que todo estacionamento terá um endereço. Define-se esta relação pela chave estrangeira *id_endereco* supramencionada.

O objetivo desta seção foi especificar o modelo de dados, a fim de expor as entidades, seus atributos e seus relacionamentos presentes na base de dados que será consultada no *Web Service*, que por sua vez, enviará estas informações como resposta ao cliente, usuário da aplicação *Android* do *Find* Parking Place*. Detalhes do Desenvolvimento da aplicação *Android* do sistema proposto serão explicados na seção a seguir.

X. Aplicação

A fim de validar a arquitetura do sistema proposto foi construída uma aplicação *Android* denominada de *Find* Parking Place*, descrita a seguir.

A. Plataforma Android

A busca dos usuários por celulares com uma interface mais sofisticada de fácil navegabilidade e com uma infinidade de recursos, inerente a procura de organizações e desenvolvedores por uma plataforma inovadora para construção de aplicações corporativas, para solucionar problemas relacionados aos negócios, bem como adquirir lucros, serviu de alicerce para a construção do *Android* pela *Google* [24].

O *Android* é uma plataforma para dispositivos móveis baseada no sistema operacional Linux, que contém várias aplicações já instaladas, assim como uma ambiente flexível para o desenvolvimento de novas aplicações. Empresas lideradas pela *Google*, tais como, HTC, LG, *Motorola*, *Samsung*, *Intel*, entre outras, do âmbito de telefonia celular, criaram um grupo denominado de *Open Handset Alliance* (OHC), no intuito de padronizar o desenvolvimento das aplicações para dispositivos móveis nesta plataforma e satisfazer as exigências do mercado [24].

No *Android* é possível utilizar a linguagem Java para o desenvolvimento das aplicações mencionadas e assim usufruir de seus recursos. Neste sentido, foi construída a aplicação *Find* Parking Place* neste projeto, sobre *Integrated Development Environment* (IDE) eclipse, através do *Android Developer Tools* (ADT), ferramenta que serve de alicerce para construção de aplicações nesta linguagem.

A subseção seguinte expõe as principais classes da plataforma *Android* para construir o *Find* Parking Place* e o funcionamento do mesmo respectivamente.

B. Principais classes

1) Activity: A classe *Activity* define a existência de uma tela na aplicação semelhantemente a classe *JFrame* do *Swing*, controla o estado desta tela, bem como a passagem de parâmetros para outras telas e métodos que serão utilizados, por exemplo, no clique de um determinado botão, entre outros [24].

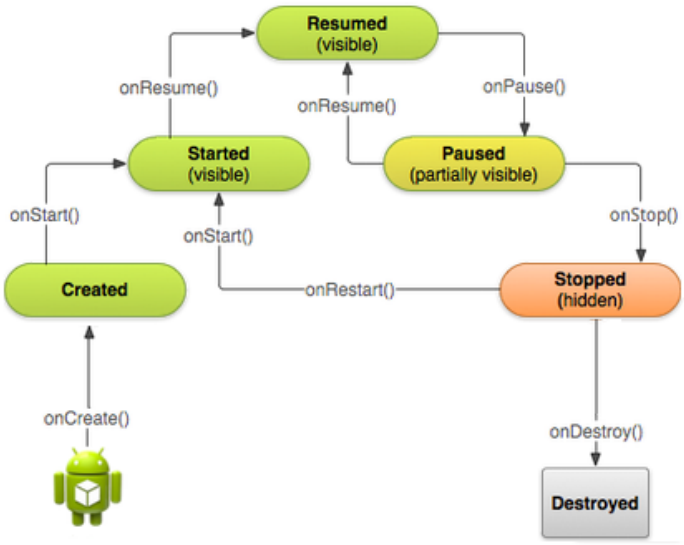

Figura 15. Ciclo de Vida da Activity [25]

Exibe-se na figura 15, o ciclo de vida da *Activity Android*, o qual é composto pelos métodos descritos sequencialmente, segundo [24].

- *onCreate(bundle)* - Método obrigatório que é acionado uma única vez. Neste método, após a criação de uma *view*, aciona-se o método *setContView(view)*, que exibe determinada *view* na tela. Posteriormente, o *onCreate(bundle)* é finalizado e o método *onStart()* é acionado, neste momento é iniciado o ciclo de vida para uma *activity* visível.
- *onStart()* - Quando a *activity* contém uma *view* e está prestes a ficar visível ao usuário, este método é acionado. O *onStart()* pode ser chamado quando o onCreate() ou onRestart() forem for finalizado.
- *onRestart()* - Acionado quando a tela da aplicação é parada temporariamente e iniciada outra vez. Este método aciona o método *onStart()* de automaticamente.
- *onResume()* - Chamado depois do *onStart()* este método representa uma determinada *activity* em execução, ou seja que está no topo da pilha.
- *onPouse()* - Caso, por exemplo, um *smartphone* entre em modo de espera no mesmo momento em que sua aplicação esteja sendo executada, esta poderá ser interrompida. Este método também é acionado para que o estado da aplicação seja salvo, no intuito de recuperá-lo depois, se necessário, através do método *onResume()*.
- *onStop()* - Método chamado caso a *activity* esteja sendo encerrada, por exemplo, quando uma tela fica invisível para o usuário. Se esta mesma tela voltar a ser invocada, o método *onRestart()* é acionado, senão se o esta *activity* ficar por um longo período parada na pilha de execução, e o SO do *Android* precisar liberar memória, o método *onDestroy* é chamado para remover esta *activity* da pilha.
- *onDestroy()* - Invocado tanto pelo SO, para liberar recursos de memória, como pela aplicação através do método

finish(), o *onDestroy* remove por completo a *activity* da pilha de execução, encerrando seu processo.

2) View: Consiste nos elementos gráficos expostos nas telas das aplicações *Android*, os quais podem ser simples, tais como, Botão, *checkbox* e imagens, ou complexos, como gerenciadores de leiaute que definem como os dados serão organizados na tela [24].

3) Intent: Segundo [24], esta classe é o núcleo do *Android*, a qual representa o envio de uma requisição em *broadcast*, isto é, uma intenção da aplicação para o sistema operacional a fim de realizar determinadas tarefas que serão executada pelo mesmo. Exemplos destas tarefas podem ser:

- Abrir uma nova tela da aplicação
- Enviar uma informação para que o sistema execute um processo em segundo plano ou um serviço.
- Exibir mapa com rota.

4) MapView: Principal classe do *Google Maps Api*, que permite a manipulação dos dados do mapa através de métodos de classes, e também fornece elementos de interface para que os usuários possam controlar o mapa da mesma forma, por exemplo, que estivesse utilizando o serviço do *Google Maps* por meio de um *browser* de Internet [26].

5) AsyncTask: Utilizada para o desenvolvimento de tarefas que serão executadas, de forma assíncrona, ou seja, separadamente da tarefa principal. Por exemplo, um usuário enquanto observa um mapa em uma aplicação de estacionamento, como as mencionadas na seção VI, poderá, por meio de uma classe *AsyncTask*, receber informações dos estacionamentos obtidas de um servidor de aplicação [27].

6) Service: Classe utilizada para executar um processo demorado em segundo plano, que geralmente consome altos recursos de memória e CPU, conhecida como serviço. Esta classe pode ser utilizada na criação de um *MP3 player* ou fazer um *download* de uma imagem na Internet, de forma transparente ao usuário [24].

7) Location: Consiste em uma classe que representa uma localização geográfica, ou seja, através dela é possível obter informação de latitude e longitude de um determinado endereço [28].

C. Funcionamento do Find* Parking Place

A partir do momento que o usuário inicia a aplicação, o *Find* Parking Place* busca dados referentes a localização deste usuário, através do *Global Positioning System* (GPS), do *smartphone* do mesmo.

Dentre as informações de localização obtidas, é utilizado o bairro para iniciar uma pesquisa via *web service* dos estacionamentos no bairro onde tal usuário se encontra. Após tal pesquisa é apresentada a tela exposta na Figura 16.

A tela inicial da aplicação apresentada na Figura 16 expõe uma mensagem de notificação da aplicação ao usuário caso a mesma não tenha encontrado nenhum estacionamento através do *web service* de acordo com a informação do bairro obtida através do GPS(como já mencionado anteriormente).

Após a apresentação desta mensagem é exibido o complemento desta tela inicial ilustrado na Figura 17 que mostra um

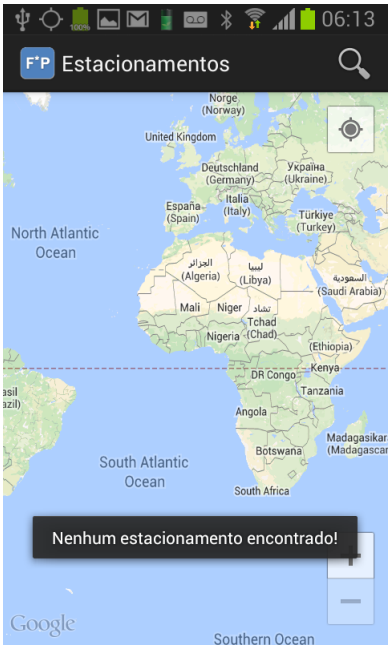

Figura 16. Tela Inicial do Find* Parking Place - Notificação

mapa destacando apenas a posição do usuário sem nenhum estacionamento, composto de uma campo no cabeçalho da tela

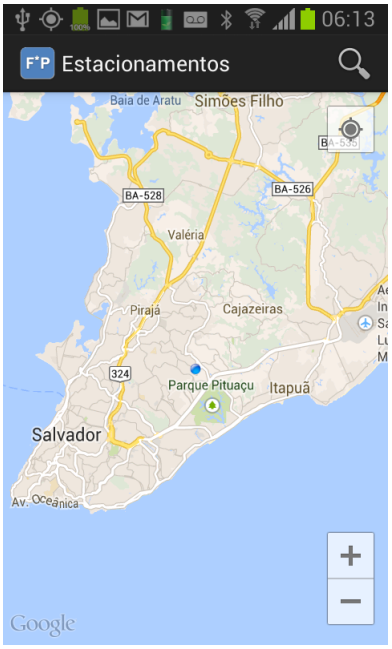

Figura 17. Tela Inicial do Find* Parking Place

para pesquisa, o qual usuário insere um determinado bairro para obter informações dos estacionamentos e suas vagas disponíveis.

Na Figura 18 é demostrada a tela de estacionamentos encontrados através da pesquisa pelo bairro Calçada da cidade de Salvador, por meio do *web service* desenvolvido neste projeto, tais estacionamentos são sinalizados no mapa por

Figura 18. Tela de Estacionamentos Encontrados

meio de marcadores, um em azul e outro em vermelho. O

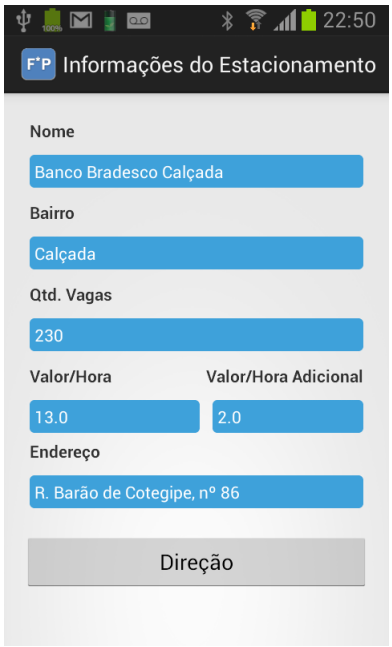

Figura 19. Tela de informações do estacionamento

marcador em azul indica que o estacionamento encontrado contem vagas, e por sua vez, o vermelho e indica a ausência de vagas no estacionamento.

Ao clicar no estacionamento que contém vaga, o usuário é direcionado para uma tela onde são evidenciadas as informações deste estacionamento, através dos campos: **Nome**; **Bairro**, **Qtd. de Vagas**; **Valor/Hora**; **Valor/Hora Adicional**; **Endereço**; como pode ser observado na Figura 19. Esta tela tem a principal função de informar a cada minuto a quantidade de vagas atualizadas do estacionamento ao usuário.

Por fim, além dos campos e função supramencionados, a tela demonstrada na Figura 19, exibe um botão de nome Direção, que quando clicado, exibe uma tela (Figura 20) indicando a rota entre a localização do usuário e o estacionamento.

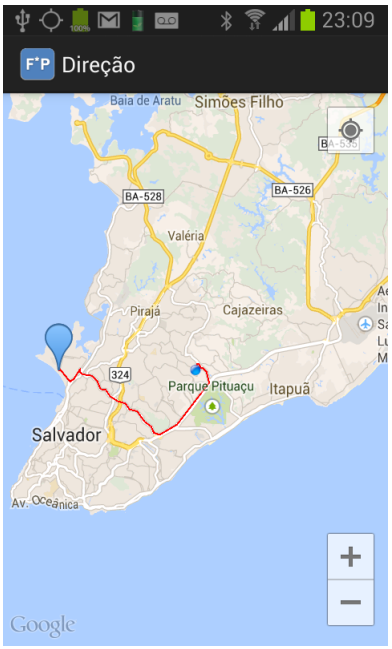

Figura 20. Rota entre Usuário e Estacionamento

A fim de ilustrar mais detalhadamente o funcionamento do *Find* Parking Place*[4], são exibidos os diagramas de sequência a seguir.

Inicia-se o Fluxo I do *Find* Parking Place* (Figura 21) no momento em que o usuário abri o sistema e o método *onCreate()* do objeto *Parkings* é acionado. Posteriormente, no método *currentUserLocation()*, por meio do GPS, o sistema obtém o bairro onde o usuário está localizado.

Após a obter o bairro, o método *accessWebService()* é invocado, o qual aciona o objeto *WebServiceClientTask*, através do método *execute(URL)*. A URL é descrita da seguinte forma: http://IP_DO_SERVIDOR:PORTA/webService/estacionamentos/bairro={P_NEIGHBORHOOD}; onde o parâmetro *P_NEIGHBORHOOD* é o bairro obtido no método *currentUserLocation()*, supramencionado.

Em seguida o método *doInBackground(String... params)* recebe a *URL* como parâmetro (*params*), e inicia o acesso ao *Web Service* do sistema (ver seção XI), através da *url* e bairro contidos em *params*. Após receber o resultado, como reposta a requisição HTTP feita ao *web service*, o método *onPostExecute(String[] result)* é iniciado.

O *onPostExecute(String[] result)* valida o resultado (*result*), enviado pelo método *doInBackground(String... params)*, e

[4]Vale salientar que no momento em que se iniciou o desenvolvimento do aplicativo *Find* Parking Place*, a versão mais atual do *andoroid* era a 4.1 *Jelly Bean*.

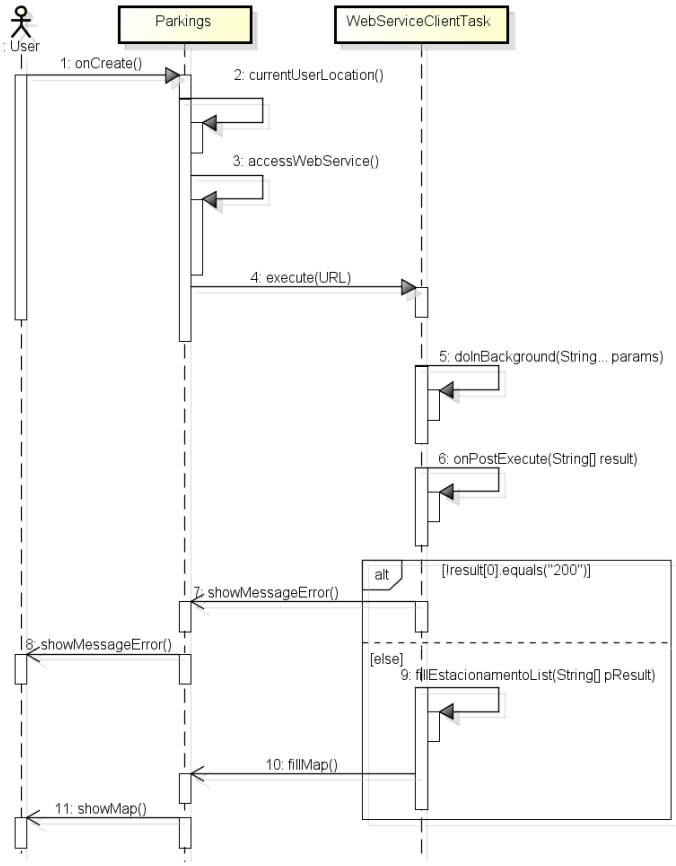

Figura 21. Diagrama de Sequência do Find* Parking Place - Fluxo I

caso este seja vazio, uma mensagem de estacionamento não encontrado é informada na tela (Figura 16). Caso contrário, o método *fillEstacionamentoList(String[ ] pResult)* é acionado para preencher a lista de estacionamento, por meio da lista em JSON contida em *pResult*. Sequencialmente, o método *fillMap()* é acionado, o qual é responsável por preencher os marcadores em azul (vaga disponível), ou vermelho, (vaga indisponível) no mapa e exibi-lo (*showMap()*) como mostra a Figura 18.

Diante dos marcadores, o usuário pode escolher em fazer uma nova busca pelo campo de pesquisa no cabeçalho da tela, o que acarretará na chamada ao método *accessWebService()* novamente, e por conseguinte em todo processo de acesso ao *web service* já mencionado, ou clicar em um dos marcadores representantes dos estacionamentos, para obter as informações dos mesmos, dando início ao Fluxo II do Sistema (Figura 22).

Ao clicar em um dos marcadores o objeto *Parkings*, por meio do método *onMarkerClick(Marker marker)* inicia o objeto *InformationParkingActivity*, enviando para a mesmo informações do estacionamento clicado.

O objeto *InformationParkingActivity* recebe a informação do estacionamento supramencionado e a envia por parâmetro(estacionamento), para o método *fillInformationParking(Estacionamento est)*, que por sua vez, usa o parâmetro enviado para exibir a tela com as informações do estacionamento (Figura 19). Em seguida, esta classe ativa o serviço, enviando como parâmetro os dados do estacionamento que são exibidos na tela, bem como, um *resultReceiver*.

O serviço é ativado através do objeto *InformationParkingService*, que inicia com o método *onStartCommand(Intent intent, int flags, int startId)*, onde é obtido os dados do estacionamento enviado pelo *InformationParkingActivity*. Neste mesmo método uma *thread* aciona o método *run()*, que utiliza essas informações do estacionamento para obter novas atualizações e o *resultReceiver* para enviar a quantidade de vagas atualizadas por minuto, a tela de informações do estacionamento (Figura 19).

Caso o usuário opte por clicar no botão direção, o método *onClick(View view)* do *InformationParkingActivity* é acionado, que por sua vez inicia o objeto *MapDirection*, o qual recebe como parâmetro o estacionamento. Nesse momento, o método *fillDirection()* inicia o objeto *DowloadTask*, através do método assíncrono *execute(URL)*.

Em seguida, o método *doInBackground (String... pUrl)* é executado, onde recebe do *execute()*, o parâmetro (*pUrl*), que contém a informação de acesso ao servidor *Google Directions* e os parâmetros de latitude e longitude do estacionamento e do usuário.

Nesse método é obtido o calculo da distância entre o estacionamento e o usuário através do JSON, que é informado ao objeto *ParserTask* através da interação entres os métodos *execute(String result)* e *doInBackground (String... pJsonData)*. No objeto *ParserTask*, o cálculo supracitado é transformado em rota, que é posteriormente enviada para o *MapDirection*, que por sua vez, a adiciona no mapa (*addPolyline(PolylineOptions lineOptions)*).

Após ter adicionado a rota no mapa, o *MapDrirection* aciona o *showDirection()* para exibir a tela de rota entre usuário e estacionamento (Figura 20).

XI. Web Service

Para validar a arquitetura deste projeto por completa, além do desenvolvimento do modelo de dados visto na seção IX, e do *Find* Parking Place* visto na seção X, também foi construído um *web service*, onde os principais características são expostas nas subseções a seguir.

A. Definição e Tecnologias utilizadas

O *web service* consiste em um serviço que visa integrar aplicações heterogêneas, onde diferentes tipos de arquiteturas podem ser utilizadas para a implementação. No projeto proposto escolheu-se a arquitetura *RESTful* pela sua simplicidade de implementação em relação aos outros que usualmente utilizam SOAP [30]. O *RESTFul* segue os princípios do *Representational State Transfer* (REST), apresentado por [31], que tem como objetivo fazer com que clientes e servidores se comuniquem na Web através do envio e recebimento de recursos por meio de *Uniform Resource Identifier* (URI) [32]. Sistemas como *Twitter* e *Flickr* utilizam estes princípios em suas *Application Programming Interfaces* (APIs).

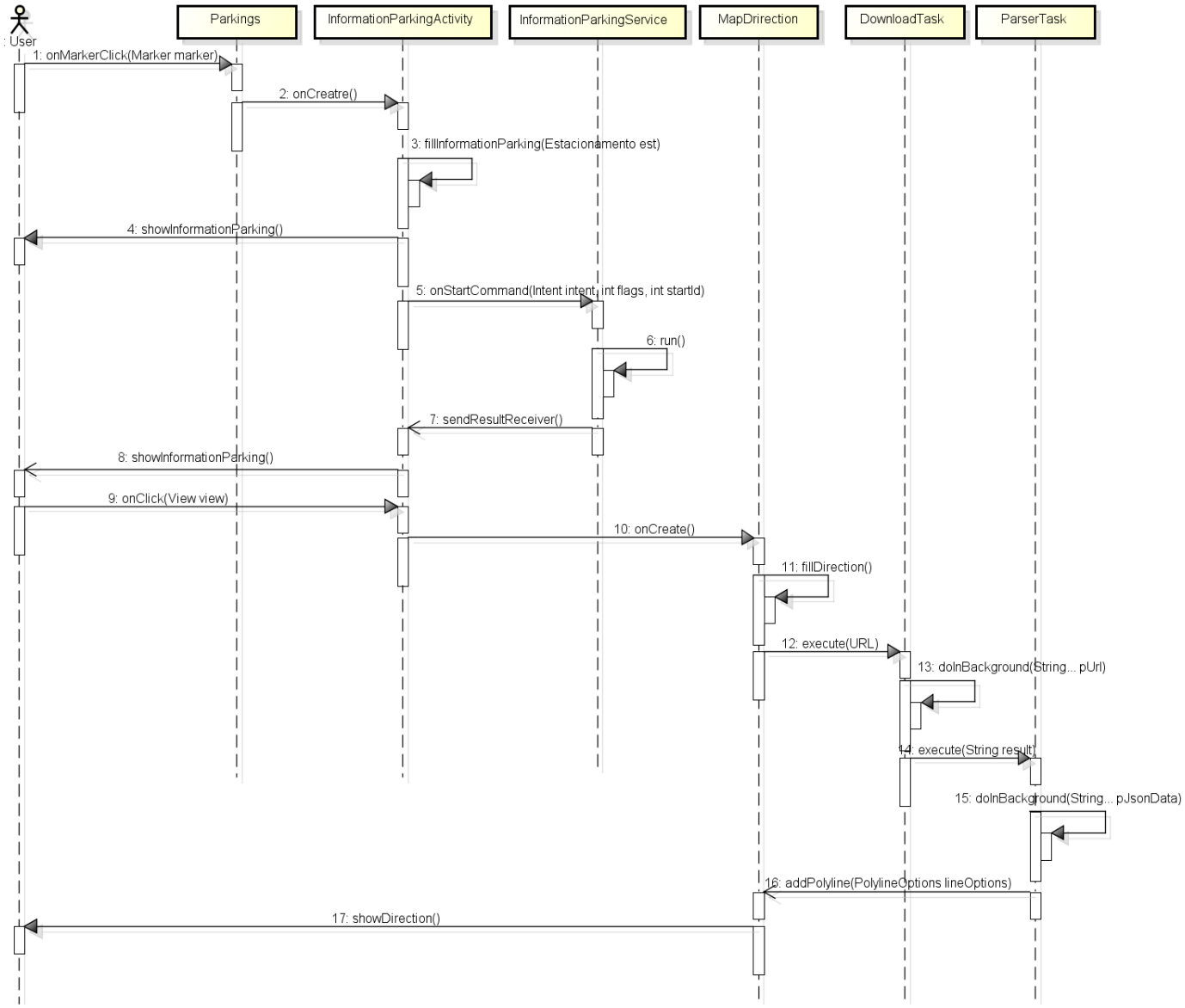

Figura 22. Diagrama de Sequência do Find* Parking Place - Fluxo II

Desenvolveu-se o *web service* deste projeto na IDE Eclipse Juno[5], em Java EE, seguindo a especificação *JAX-RS* que é integrante desta plataforma. Esta especificação determina a criação de *web services* seguindo a arquitetura *RESTful*, através de anotações que ocultam a complexidade de configurações extras, implementada através do *Jersey*, um software de código aberto que é a referência indicada pela *Oracle* para implementar o *JAX-RS* [33]. Estas anotações são apresentas a seguir:

- *@GET* - Define o retorno de algum recurso como, XML ou JSON, pelo método. Onde o primeiro é uma linguagem de marcação de texto usada para compartilhar informações estruturadas, e o segundo é uma notação textual baseada na linguagem *Java Script*, usada também para compartilhar informações estruturadas. Por ser textual, e por isso mais leve para trafegar sobre a rede, foi utilizado JSON no desenvolvimento deste web service;
- *@Path* - Recebe uma *String* como parâmetro que indica o endereço da URL informada pelo usuário como já citado;
- *@Produces* - Define o formato que será retornado do *web service* proposto, que será o JSON, como descrito anteriormente;
- *@PathParam* - Refere-se ao parâmetro da URL que será informado via HTTP por um determinado cliente. Neste *web service* é utilizado o parâmetro *P_NEIGHBORHOOD*, que indica o bairro onde o usuário está ou onde ele queira estacionar.

Com o objetivo de facilitar o acesso a base de dados pelo *web service*, que por sua vez retorna uma lista de estacio-

[5]http://www.eclipse.org/juno/

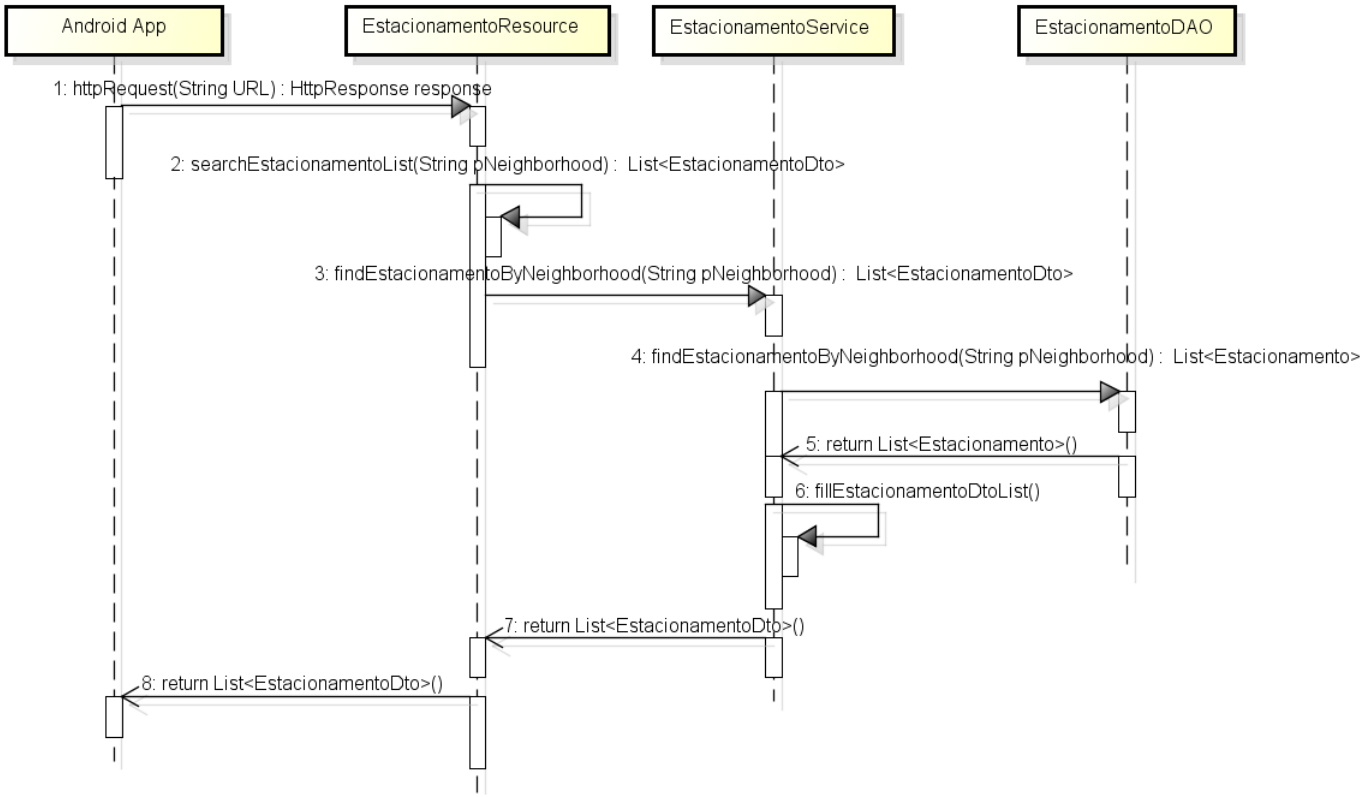

Figura 23. Diagrama de Sequência do Web Service

namentos para o cliente, também foi utilizado o *Hibernate* junto com a especificação *JPA*. O *Hibernate* é um *framework* de mapeamento objeto-relacional [34] que implementa a especificação JPA para persistência dos dados nesta base. Este mapeamento é dado por anotações sobre os campos dos atributos das classes, que representam os campos das entidades na base de dados, vista na seção IX. As principais anotações utilizadas neste *web service* são descritas a seguir:

- *@id* - Define a chave primária da tabela;
- *@Column* - Define o campo da tabela;
- *@OneToOne* - Referencia um relacionamento de um para um. Ou seja, informa que um estacionamento só poderá ter no mínimo um e no máximo um endereço, e uma posição bem como, um endereço e uma posição só poderão pertencer a um estacionamento;
- *@JoinColumn* - Referencia os campos *id_endereco* e *id_posicao* que são chaves estrangeiras para as tabelas *endereco* e *posicao*, também detalhadas na seção IX.

Para executar o *web service* foi utilizado o *web container Tocamt*, entretanto, para uma versão comercial será utilizado um servidor de aplicação como *JBoss* [6].

B. Funcionamento do Web Service

A fim de ilustrar o funcionamento do *web service* do sistema proposto, exibe-se na Figura 23 o seu diagrama de sequência.

[6]https://www.jboss.org

O fluxo de execução do *web service* é iniciado por meio de uma requisição (*httpRequest (String URL)*) da aplicação *android* deste projeto. Essa requisição é recebida pelo objeto *EstacionamentoResource* através do método *searchEstacionamentoList(String pNeighborhood)*, onde o (*pNeighborhood*), é o bairro informado como parâmetro contido nesta URL.

Em seguida, o objeto *EstacionamentoService* é acionado através do método *findEstacionamentoByNeighborhood(String pNeighborhood)*, que por sua vez, faz uma chamada a um outro método de mesma assinatura, porém do objeto *EstacionamentoDAO*.

No *EstacionamentoDAO* através do método supracitado, realiza-se a consulta dos estacionamentos de acordo com o bairro indicado. O resultado dessa consulta é retornado ao objeto *EstacionamentoService* (*return List<Estacionamento>*), que invoca o método *fillEstacionamentoDtoList()*.

O *fillEstacionamentoDtoList()* é responsável por transformar a lista de estacionamento obtida, através do preenchimento de uma outra, do tipo *Data Transfer Object* (DTO). Desta forma, cada elemento da lista de estacionamento composto da entidade *Estacionamento* e suas composições, as entidades *Endereco* e *Posicao* (ver Seção IX), são transferidos para uma única entidade, facilitando o processo de transformação desta lista unificada, para uma outra em formato JSON.

Posteriormente, uma lista de estacionamento DTO (*return List<EstacionamentoDto>*) é retornada para o objeto *EstacionamentoResource*, pelo *EstacionamentoService*, que responde

a requisição do usuário enviando a lista de *EstacionamentoDTO*, em formato JSON, finalizando o processo de consulta a base de dados através do *Web Service* proposto.

XII. Ambiente

Considerou-se os seguintes aspectos como forma de validação do sistema proposto:

- *Configuração do Web Service* - Para retornar os estacionamentos para o *smartphone* contendo o *Find* Parking Place* no tempo de reposta mencionado, foi utilizado as seguintes tecnologias para o desenvolvimento e execução do *web service*:
  1) Na área de servidores e *web containers* - *Tomcat*, na versão 7.4.1. Utilizou-se este *container* pela sua facilidade de configuração frente aos servidores de aplicação como *JBoss*.
  2) No ramo de tecnologias para o desenvolvimento de *web services* - *RESTful*, pela sua simplicidade de implementação em relação aos outros que usualmente utilizam SOAP [30].
  3) Dentre as IDEs para desevolvimento java - Eclipse versão Juno.
  4) Dentre as tenologias para envio de informações via *web service* - JSON, por ser mais leve que XML para trafegar na rede.
- *Sistema Gerenciador de Banco de Dados (SGBD)* - O *PostgreSQL* foi escolhido por ser gratuito e composto de funcionalidades tão sofisticadas quanto as de um SGBD proprietário.
- *Tempo de Resposta do Web Service* - O tempo que o *web service* gasta para consultar os estacionamentos na base de dados é menor que 3 segundos.
- *Teste da Aplicação* - Aplicação foi testada em um *smartphone* com sistema operacional *Android* versão 4.1.

Com base nestes aspectos pôde-se constatar que os dados armazenados na base de dados requisitados através da aplicação *Find* Parking PLace*, instalada no dispositivo móvel com versão de *Android* 4.1, foram retornadas de forma correta pelo *web service*, que por sua vez, obteve tais dados do SGBD *PostgresSQL* desenvolvido neste trabalho, em um tempo menor que 3 segundos.

Como resultado da análise foram obtidos os resultados:

- A aplicação apresentou de forma correta os dados do banco de dados.
- Em alguns pontos a aplicação se apresentou transparente para o usuário, contemplando questões de ubiquidade, as quais podem ser destacadas:
  1) *Invisibilidade* - Ao iniciar a aplicação *Find* Parking Place*, o usuário pode receber informações dos estacionamentos do bairro onde está localizado, sem que o mesmo precise informar algum parâmetro para a aplicação.
  2) *Heterogeneidade* - O modelo foi criado para interligar ambientes e tecnologias diferentes, como a aplicação sendo executada em *smartphone* de um determinado usuário, interligada por um *web service*, executado em um servidor de aplicação, a uma base dados mantida por um SGBD.
- O projeto como todo se mostrou distribuído visto, que funcionou corretamente por meio de mensagens entre os três artefatos que compõem este trabalho.
- Aplicação não teve nenhum comportamento estranho por está instalada em um dispositivo móvel real, satisfazendo as questões em torno da mobilidade computacional.
- A aplicação ainda necessita de maior aprimoramento no que diz respeito a invisibilidade e processamento. Apesar de apresentar informações do bairro sem que o usuário tenha que interagir com o aplicação, o modelo não suporta em seu estado atual mecanismos para sempre exibir as informações do estacionamentos a medida que o usuário mude de localização. E também ainda contem um processamento elevado do lado do cliente para exibir tais estacionamentos, bem como o numero de vagas dos mesmos.

XIII. Resultados e Conclusão

Um dos principais objetivos das aplicações ubíquas é prover serviços a usuários com um grau de conhecimento específico sobre o contexto que estão inseridos. Sendo assim, tais aplicações podem auxiliar na prevenção e resolução de problemas cotidianos enfrentados por estes usuários. Um exemplo destes problemas são os relacionados ao transporte, tais como, congestionamento e estacionamento dos grandes centros urbanos do Brasil, como Salvador, devido ao aumento da quantidade de carros e a falta de infraestrutura nestas cidades para suportar este aumento, principalmente em datas comemorativas.

A fim de auxiliar os usuários a encontrar vagas disponíveis nos estacionamentos, foram desenvolvidos aplicativos como *BestParking*, *OndeParar*, *ParkLocator*, entre outros, os quais informam vagas disponíveis, endereço, valor hora, dentre outras informações dos estacionamentos. Entretanto, observou-se que tais aplicativos não fornecem mecanismos para que o usuário possa saber a quantidade de vagas atualizadas dos estacionamentos por minuto.

No intuito de proporcionar aos usuários através de seus dispositivos móveis conectados a rede *wireless*, localizar os estacionamentos de centros e estabelecimentos comerciais de centros urbanos, foi desenvolvido neste trabalho um sistema ubíquo para gerenciar tais estacionamentos denominado de *Find* Parking Place*.

Este sistema visa contribuir para o crescimento do desenvolvimento de sistemas móveis e distribuídos a fim de torná-los mais ubíquos para solucionar problemas do cotidianos dos usuários, tais como encontrar estacionamentos.

Constatou-se que a arquitetura desenvolvida neste trabalho possui características semelhantes arquitetura proposta por Fiorini [4] tais como, a utilização do modelo cliente-servidor, foco no lado do cliente por meio de dispositivos móveis, interação com ambiente externos entre outros. A arquitetura de Fiorini, se mostra mais independente da arquitetura deste projeto, pois tem todo o processamento do lado do servidor, ficando

a cargo dos distintos clientes apenas consumir. Entretanto a arquitetura deste projeto é pouco independente em relação modelo a esse outro modelo, pois esta voltada atualmente apenas para dispositivos que utilizam o Sistema Operacional *Android*, porém, ganha na construção de aplicações mais ricas através dos recursos oferecido por esse S.O.

Observou-se que a arquitetura proposta por Carmo [5], a qual serviu de referência para o modelo arquitetural deste trabalho, pois além de ser voltado para a localização de estacionamentos em centros urbanos, faz uso de SBL (Serviço Baseado em Localização), interação com ambientes externos através de *web service* e tem como base a plataforma *android*, não possui mecanismos para exibição de vagas atualizadas em um determinado tempo, nem para exibição de rotas entre a localização do usuário e o estacionamento pretendido, apresentados na arquitetura deste trabalho. Todavia, o modelo deste projeto não apresenta mecanismos para emitir mensagem de voz nem alertas sonoros apresentada na arquitetura de Carmo.

Como forma de validação do sistema proposto, foram construídos três artefatos de software: (i) Modelo de Dados; (ii) *web service*; (iii) aplicação android; os quais contemplaram as principais funcionalidades do sistema para localizar informações dos estacionamentos de centros e estabelecimentos comerciais, baseados em preço, hora, endereço, quantidade de vagas, entre outros.

Entretanto, o sistema proposto, precisa de um maior aprimoramento no que diz respeito a invisibilidade e desempenho, pois apesar de apresentar informações do bairro sem que o usuário tenha que interagir com o aplicação, o *Find* Parking Place* não suporta em seu estado atual mecanismos para sempre exibir as informações do estacionamentos à medida que o usuário mude de localização. E também ainda contém um processamento elevado do lado do cliente para exibir tais estacionamentos, bem como o número de vagas dos mesmos.

Para trabalhos futuros sugere-se a adaptação do sistema para que possa ser executado em outros Sistemas Operacionais como IOs. Sugere-se também o desenvolvimento de mecanismos para que usuário receba informações dos estacionamentos na base de dados mesmo que mudem de localização, sem precisar interagir com o dispositivo móvel. Recomenda-se também o uso de mecanismos para minimizar o uso de processamento do lado cliente, bem como a utilização de testes mais rebuscados em relação a maior número de usuários com seus *smartphones* e uma maior quantidade de base dados interagindo com *web service*.

XIV. Agradecimentos

Primeiramente gostaria de agradecer a Deus por sempre iluminar meus caminhos, a Jesus Cristo, que sempre está comigo principalmente nos momentos mais difíceis e a Maria Santíssima, pela sua interseção e por não deixar-me fraquejar diante das adversidades.

Agradeço a minha mãe Rita Mota, minha fortaleza, sem ela nada disso seria possível. A meu irmão Melqui Mota, as minhas madrinhas Luciana Mota e Izabel, minha tia Lucília e familiares pelo apoio.

Um agradecimento especial a minha orientadora professora Msc. Flávia Maristela Nascimento, pela paciência, dedicação e incentivo para que a conclusão deste trabalho fosse possível. Ao professor Dr. Eduardo Manuel de Freitas Jorge, pelos incentivos relacionados ao ingresso e permanência na especialização. Ao professor Dr. Frederico Jorge Barbosa pelas revisões do trabalho.

A todos os Mestres e Doutores da Especialização em Computação Distribuída e Ubíqua, dos quais os ensinamentos serviram de base para construção deste projeto.

Aos colegas da Especialização em Computação Distribuída e Ubíqua. A meus amigos José A. Roberto Ferraz, Cássio Oliveira e Jaqueline Mota pelo apoio ao decorrer do curso, e a todos os outros que de forma direta ou indireta contribuíram para a conclusão de mais esta etapa na minha vida.

Referências

[1] Krumm, J. *Ubiquitous computing fundamentals*. Boca Raton: Chapman & Hall/CRC Press, 2010.

[2] Araujo, R. B. *Computação Ubíqua: Princípios, Tecnologias e Desafios*. XXI Simpósio Brasileiro de Redes de Computadores. São Paulo, São Carlos: UFSCar, 2003.

[3] Weiser, M. *The Computer for the 21st Century*. Scientific American, September, 1991.

[4] Fiorini, M. *Uma Arquitetura Genérica de Software para Disponibilização de uma Aplicação Web para Dispositivos Móveis*. 2006. 97p. Dissertação (Mestrado em Engenharia Elétrica) Área de Concentração em Automação e Sistemas, Universidade Federal de Santa Catarina, Florianópolis, 2006.

[5] Carmo, R. L. C. *Modelo de Arquitetura de uma Aplicação Sensível ao Contexto para Localização de Estacionamentos em Centros Urbanos*. Universidade do Estado da Bahia, Salvador, 2013.

[6] BestParking. *BestParking App* https://play.google.com/storeetails?id=com.bestparking, Goolgle play. Acesso em: 20 Mar. 2014.

[7] OndeParar. *OndeParar App* https://play.google.com/store/apps/details?id=br.com.ondeparar, Goolgle play. Acesso em: 20 Mar. 2014.

[8] Let'sPark. *Let'sPark App* https://play.google.com/store/apps/details?id=br.com.letspark.app, Goolgle play. Acesso em: 20 Mar. 2014.

[9] Forman, G. H.; Zahorjan, J. *The Challenges of Mobile Computing*. University of Washington, April, 1994.

[10] Exame.com.*Brasil é o quarto país do mundo em número de smartphones*. http://exame.abril.com.br/tecnologia/noticias/brasil-e-o-quarto-pais-do-mundo-em-numero-de-smartphones. Acesso em: 21 Ago. 2013.

[11] Coulouris, G.; Dollinore, J.; Kindberg, T. *Sistemas distribuídos: conceitos e projeto*. 4.ed. Porto Alegre, RS: Bookman, 2007.

[12] ITU.*Measuring the Information Society*. http://www.itu.int/en/ITU-D/Statistics/Documents/publications/mis2013/MIS2013_without_Annex_4.pdf. Acesso em: 1 Mai. de 2014.

[13] A. Avizienis et al. *Basic concepts and taxonomy of dependable and secure computing*. In Dependable and Secure Computing, IEEE Transactions on Vol. 1, 2004.

[14] Super Interessante. *A caneta esferográfica*. http://super.abril.com.br/cotidiano/caneta-esferografica-438598.shtml. Acesso em: 15 Fev. 2013.

[15] Weiser, M.; Brown, J. S. *The coming age of calm technology. In: Denning, P. J., and Metcalfe, R. M. (Eds.), Beyond Calculation: The Next Fifty Years of Computing*. Springer-Verlag, New York, NY, 1997.

[16] Silva, G. K. C. *Implementação de Serviços Baseados em Localização Utilizando Arquiteturas e Padrões Abertos*. Dissertação de Mestrado Profissional. São Paulo, Campinas: UNICAMP, 2005.

[17] Schiller,j; Voisard, A. *Location-Based Services*, Morgan Kaufmann Series in Data Management Systems, Elsevier Inc., 2004.

[18] Waze. *Waze* https://www.waze.com/pt-BR. Acesso em: 10 Jan. 2014.

[19] BestParking. *BestParking App* https://itunes.apple.com/app/compare-parking-rates/id383076098?mt=8, Itunes. Acesso em: 20 Mar. 2014.

[20] OndeParar. *OndeParar App* https://itunes.apple.com/br/app/ondeparar/id718043365?mt=8, Itunes. Acesso em: 20 Mar. 2014.

[21] Let'sPark. *Let'sPark App* https://itunes.apple.com/us/app/lets-park/id695606122?mt=8, Itunes. Acesso em: 20 Mar. 2014.
[22] PgModeler. *PostgreSQL Database Modeler* http://www.pgmodeler.com.br/. Acesso em: 13 Jun. 2013
[23] PostGreSQL. *Sistema Gerenciador de Banco de Dados* http://www.postgresql.org.br/sobre. Acesso em: 13 Jun. 2013
[24] Lecheta, R. R. *Google Android: aprenda a criar aplicações para dispositivos móveis com o Android SDK*. Ed. Novatec. 2ł ed. São Paulo, 2009, 2010.
[25] Lifecycle Callbacks. *Understand the Lifecycle Callbacks* http://developer.android.com/training/basics/activity-lifecycle/starting.html#lifecycle-states. Acesso em: 05 Jan. 2014
[26] Google Maps. *API Android do Google Maps* https://developers.google.com/maps/documentation/android/. Acesso em: 15 Ago. 2013
[27] AsyncTask. *AsyncTask Class* http://developer.android.com/reference/android/os/AsyncTask.html. Acesso em: 25 Ago. 2013
[28] Location. *Location Class* http://developer.android.com/reference/android/location/Location.html. Acesso em: 25 Ago. 2013
[29] Developers. *Training, API Guides, Reference, Tools, Google Services and Samples* http://developer.android.com/develop/index.html. Acesso em: 05 Fev. 2013
[30] Macedo, A. *Criando um WebService Restful em Java* http://www.k19.com.br/artigos/criando-um-webservice-restful-em-java/. Acesso em: 23 Jun. 2013
[31] Fielding, R. T. *Architectural Styles and the Design of Networkbased Software Architectures*, California, University of California, Irvine 2000
[32] Li, H.*RESTful Web Service Frameworks in Java*. Informatization Office, Shanghai Lixin Univ. of Commerce, Shanghai, China, 2011.
[33] Dubois, J.*Introduction to JerseyŮa Standard, Open Source REST Implementation*. http://www.oracle.com/technetwork/articles/javaee/jersey-part1-159891.html. Acesso em: 27 Jun. 2013
[34] King, G. et al. *HIBERNATE - Persistência Relacional para Java Idiomático*. http://docs.jboss.org/hibernate/core/3.5/reference/pt-BR/html/index.html. Acesso em: 19 Jun. 2013