Desenvolvimento e Integração de uma Aplicação Móvel a um Sistema Distribuído: Estudo de Caso de um Sistema de Gerenciamento de Vagas

Dênis Taironi Leorne da Cunha[1], **Pedro Henrique de Sousa Brito**[1]
Emanuel Ferreira Coutinho[1]

[1] Instituto Universidade Virtual (UFC Virtual)
Universidade Federal do Ceará (UFC) – Fortaleza – CE – Brasil

denis.taironi@gmail.com,p.1392@hotmail.com,emanuel@virtual.ufc.br

Abstract. *Currently, more and more technology is inserted in our lives. This insertion can be done in many different ways, such as computers embedded in clothing labels, household appliances or light switches, and invisibly to the user. This reality depicts the scenario of ubiquitous computing. However, numerous concepts and implementation challenges in developing such systems arise with this scenario. Based on these challenges, the objective of this study was to develop a mobile application and define strategies for application integration, composing a ubiquitous distributed system. The proposed system it a manager for parking lots, which has components ranging from sensors that monitor places until servers that collect data and manage customer orders. The methodology was to develop the vacancy management system, focusing on a mobile application, integration with the system, features and functionality. The results were satisfactory. The information flow between the sensors and the final application was constant, and the mobile application integration solution has proved simple and efficient.*

Resumo. *Atualmente, cada vez mais a tecnologia está inserida em nossas vidas. Essa inserção pode ocorrer das mais diferentes maneiras, como computadores inseridos em etiquetas de roupas, utensílios domésticos ou interruptores de luz, e de forma invisível para o usuário. Essa realidade retrata o cenário da computação ubíqua. Entretanto, inúmeros conceitos e desafios de implementação no desenvolvimento de sistemas dessa natureza surgem com esse cenário. Baseado nesses desafios, o objetivo deste trabalho foi desenvolver uma aplicação móvel e definir estratégias de integração da aplicação a um sistema base proposto, compondo assim um sistema distribuído ubíquo. O sistema base proposto trata-se de um gerenciador de vagas de estacionamento, que conta com componentes que vão desde sensores que monitoram as vagas, até servidores que coletam os dados e gerenciam os pedidos dos clientes. A metodologia consistiu em desenvolver o sistema de gerenciamento de vagas, sendo o foco a aplicação móvel, sua integração com o sistema, características e funcionalidades. Os resultados obtidos foram satisfatórios. O fluxo de informação entre os sensores e as aplicações finais foi constante, e a solução de integração da aplicação móvel se mostrou simples e eficiente.*

1. Introdução

Nos tempos atuais, existem milhões de dispositivos computacionais, como *smartphones*, PCs, *notebooks*, *smartwatches* e microcontroladores em geral conectados à internet trocando dados a todo o momento. O resultado da troca de informações e integração desses dispositivos é a infinidade de aplicações possíveis que podem ser criadas com o intuito de fornecer para o usuário facilidade e funcionalidade na realização de tarefas do seu cotidiano. Como aplicativos de corrida que enviam a distância e tempo do exercício para as redes sociais, como o RunKepper[1] ou ainda aplicativos que auxiliam na solicitação de um transporte urbano como o Uber[2].

Integrar estas atividades à rede, difundindo suas informações com mais pessoas e aumentando as suas funcionalidades, faz com que esses sistemas se tornem melhores e maiores. Sistemas dessa natureza são considerados sistemas distribuídos, que, segundo Couloris [Coulouris et al. 2013], são "sistemas no qual os componentes de hardware ou software localizados em computadores interligados em rede se comunicam e coordenam suas ações apenas enviando mensagens entre si".

Sendo assim, um exemplo de sistema distribuído que oferece funcionalidades facilitadoras para atividades do dia-a-dia das pessoas seria um sistema que possibilitasse a compra de ingressos de cinemas pela internet, com a facilidade de visualizar os assentos disponíveis na sessão e reservá-los previamente, como em ingresso.com[3], ou ainda aplicativos de GPS que utilizam a velocidade média dos usuários para determinar a situação do trafego de veículos, como o Waze[4].

Nesse cenário de sistemas distribuídos nos deparamos com o conceito de computação ubíqua. A computação ubíqua surge então da necessidade de se integrar mobilidade com a funcionalidade da computação pervasiva, ou seja, qualquer dispositivo computacional, enquanto em movimento conosco, pode construir, dinamicamente, modelos computacionais dos ambientes nos quais nos movemos e configurar seus serviços dependendo da necessidade [de Araujo 2003].

Desta maneira, entende-se que o paradigma de computação ubíqua é uma junção entre a computação móvel, paradigma que está ligado ao conceito de mobilidade dos recursos computacionais, e a computação pervasiva, que diz que os computadores estão embarcados ao ambiente tornando-se invisíveis ao usuário.

Assim exposto, o paradigma de computação ubíqua se mostra uma abordagem poderosa no desenvolvimento de sistemas distribuídos. E envolvendo uma grande diversidade de conceitos e tecnologias com o intuito de superar desafios como infraestrutura, adaptabilidade, segurança, sensibilidade ao contexto, descoberta de serviços, escalabilidade e consumo de energia. Desta maneira, torna-se de crucial importância construir estratégias de desenvolvimento com a finalidade de experimentar conceitos e tecnologias que abordem esses desafios na construção de sistemas ubíquos.

Sob esta ótica, quando olhamos para um estacionamento, serviço usado diariamente por inúmeras pessoas, observamos um cenário rico em oportunidades de melhorias

[1]RUNKEEPER - http://runkeeper.com/
[2]UBER - https://www.uber.com/pt/
[3]INGRESSO.COM - http://www.ingresso.com/
[4]WAZE - http://www.waze.com/pt-BR/

desse serviço por meio de aplicações distribuídas sob o viés da computação ubíqua. Por que não podemos visualizar a ocupação dos setores pelos nossos dispositivos? Por que não há uma opção de sermos avisados caso ocorra algum incidente com o carro no estacionamento? Por que a máquina de tíquete na entrada do estacionamento não pode sugerir um setor mais vago para ajudar o motorista a estacionar? Por que todas as informações estão presas dentro do ambiente e são apenas transmitidas por lâmpadas verdes e vermelhas?

Registrar as informações dos sensores no estacionamento e transmiti-las para os clientes pode se tornar um problema. Os sensores, os servidores e os aplicativos finais precisam se comunicar com perfeição, independente dos sistemas operacionais e dos dispositivos nos quais executam.

Para resolver esse problema é necessário criar estratégias para integrar todas estas partes diferentes do sistema, desde a comunicação dos sensores com as aplicações servidoras até a comunicação com os diversos dispositivos finais dos usuários. Estas estratégias necessitam ser simples e possíveis de serem implementadas nos sistemas já existentes, para que se tornem atraentes a aplicações reais.

Assim sendo, o presente trabalho busca, mediante estudo de caso do sistema de gerenciamento de vagas de estacionamento, proporcionar estratégias de integração de uma aplicação móvel a um sistema distribuído. Os objetivos secundários do trabalho são: desenvolver uma forma de registrar as informações dos sensores; criar um protocolo de comunicação entre as partes do sistema; criar um sistema de comunicação multiplataforma, permitindo adicionar novas partes ao sistema com maior facilidade; e expor funcionalidades pertinentes na aplicação móvel referente às problemáticas que cercam um sistema de gerenciamento de vagas de estacionamento.

A metodologia do trabalho consiste na construção de um sistema de gerenciamento de vagas de estacionamento, sendo o foco do trabalho a construção de uma aplicação móvel e sua integração ao sistema base. Serão abordadas questões como: as formas de comunicação entre a aplicação e o sistema base; protocolos estabelecidos para troca de informações; características de construção da aplicação como telas e recursos da plataforma de desenvolvimento; e funcionalidades pertinentes a problemática exposta, no caso uma gerência de vagas de estacionamento. O sistema base citado trata-se de uma aplicação servidora com um banco de dados que armazenará informações coletadas pelos sensores de presença (componentes embarcados do sistema).

Este trabalho contribui com uma arquitetura simples, de fácil manutenção e que utilize características básicas de programação para integrar diferentes dispositivos, expondo, assim, as possibilidades que os sistemas de estacionamento comuns teriam caso fossem conectados à rede e combinados com diversos outros dispositivos.

Neste trabalho será demonstrado como foi construído o sistema que atende aos objetivos propostos. Na primeira seção, serão apresentados uma visão do sistema criado para contextualizar a problemática exposta, os componentes e a forma que foram utilizados na construção do sistema, desde sua parte física, com sensores e o Arduino, até a aplicação servidora e aplicativos clientes. Em seguida, serão descritos especificamente a construção da aplicação móvel e as estratégias tomadas para integrar essa aplicação ao sistema. Por fim, serão apresentadas a conclusão e propostas de trabalhos futuros.

2. Sistema de Gerenciamento de Vagas em Estacionamento

O sistema de gerenciamento de vagas de estacionamento oferece a funcionalidade de checagem do estado no qual se encontram. Ele é dotado de sensores que monitoram constantemente as vagas e informam a um servidor o estado ao qual se encontram (ocupadas ou livres). Essas informações são armazenadas em um banco de dados para que posteriormente sejam utilizadas por aplicações clientes que, munidas desse insumo, oferecem várias funcionalidades. A Figura 1 exibe uma visão da arquitetura do sistema, ressaltando as camadas que compõem o sistema assim como algumas tecnologias e ferramentas utilizadas na construção do mesmo.

2.1. Componente Embarcado

Para Barros e Cavalcante (2013), "um sistema embarcado ou embutido (*embedded system*) pode ser definido como um sistema computacional especializado que faz parte de uma máquina ou sistema maior" [Barros e Cavalcante 2016]. Sob essa luz, pode-se perceber recursos computacionais embarcados em diversos sistemas, como em sistemas de automóveis sensíveis à presença do motorista, em sistemas de segurança que monitoram usuários em bancos e aeroportos, ou ainda em plataformas de videogames que contam com sensores de movimento para gerar interatividade entre o usuário e o jogo.

Com o amadurecimento de conceitos como o de computação pervasiva, conceito este que propõe um mascaramento do sistema computacional em relação aos usuários, componentes como sensores, microcontroladores e microprocessadores são recursos indispensáveis no desenvolvimento desses tipos de sistemas. Isto se dá pelo fato de serem componentes em sua maioria pequenos e de boa adaptabilidade física, baixo custo, possuírem finalidades específicas e com grande poder de integração a outros sistemas.

Para um sistema de gerenciamento de vagas de estacionamento, cujo a principal funcionalidade é o constante monitoramento do estado das vagas, contar com sensores

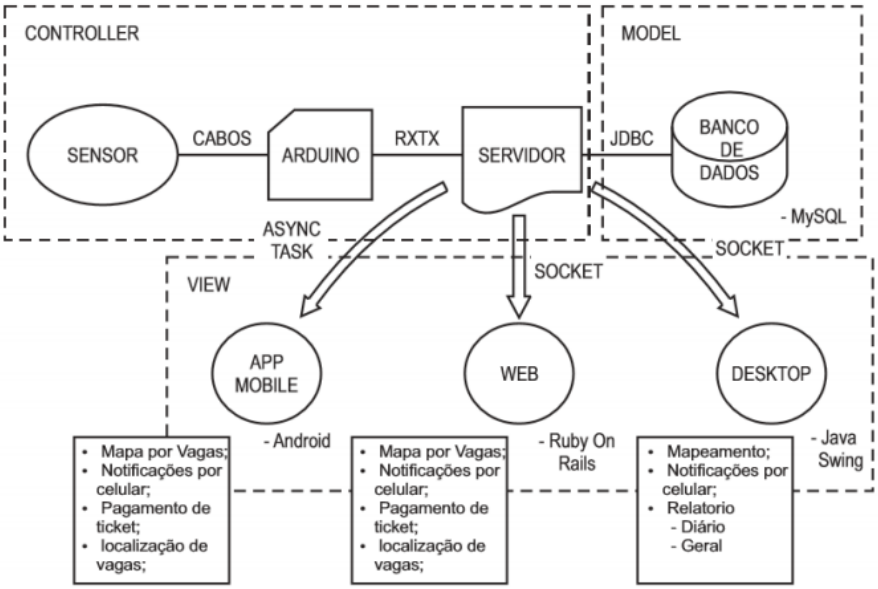


Figura 1. Arquitetura de aplicação

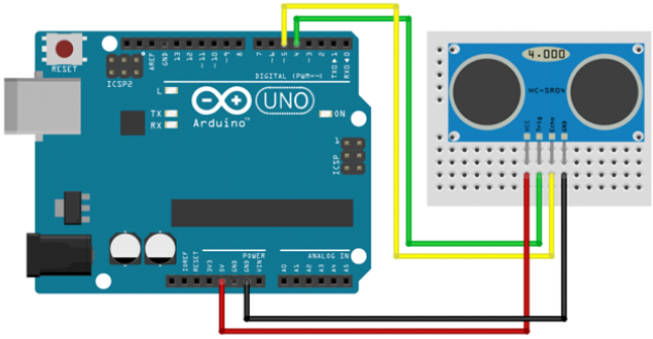


Figura 2. Esquematização de montagem dos componentes embarcados

de presença acoplados a um microprocessador se faz indispensável na realização desta tarefa. Assim, o sistema conta com sensores ultrassônicos de distância modelo HC–SR04 que por meio de uma simples *protoboard*, que se trata de uma placa de ensaio com furos e conexões condutoras para montagem de circuitos elétricos experimentais, e *jumpers*, que são pequenos condutores utilizados para conectar dois pontos de um circuito eletrônico, se conectam a um microprocessador Arduíno. A Figura 2 mostra um esquema da montagem destes componentes.

2.2. Servidor

Após o recolhimento dos dados do estacionamento, é necessário que este material seja salvo para futuro uso. O responsável por estas ações no sistema é um servidor, denominado Coletor de Dados, que possui os métodos necessários para realizar tais atividades. Também é necessário um servidor que gerencie o acesso aos dados do sistema, servindo de intermédio entre os clientes e a base de dados, chamado de aplicação servidora.

2.2.1. Coletor de Dados

O coletor de dados é o primeiro servidor a entrar em ação no sistema. A primeira função do coletor é receber do Arduíno os dados do estado das vagas do estacionamento, enviado pela conexão serial do computador. O Java não pode manipular o serial nativamente, por esta razão foi utilizado a API (*Application Programming Interface* ou Interface de Programação de Aplicação) RXTX [Rxtx 2016]. Por meio dela é possível ter o controle dos dados na serial e apontá-los a uma variável no sistema.

As informações sobre o estacionamento são salvas pelo Arduíno em uma *string*, na qual o caractere 0 (zero) representa uma vaga vazia e o caractere 1 (um) representa uma vaga ocupada, para depois serem enviadas.

De posse destas informações novas sobre o estacionamento, é necessário atualizar vários dados presentes no banco de dados para que as aplicações finais tenham sempre os dados certos. Como o estado do estacionamento pode mudar constantemente com a entrada e saída de veículos, essa atualização deve ocorrer de forma sistemática.

O Arduíno pergunta o estado dos sensores a cada 15 segundos e envia estas informações ao servidor, que depois as guarda no banco de dados. Este intervalo de tempo foi escolhido para dar ao cliente que está estacionando o veículo tempo suficiente

para manobrá-lo, sem gerar assim informações de vaga ocupada caso ele entre por alguns instantes na área de dos sensores.

Para acessar o banco de dados, e atualizá-lo, utilizou-se uma API chamada **JDBC** (*Java Database Connectivity*), responsável por executar comandos SQL no banco. O método do sistema responsável por atualizar o banco é o **Atualizar()**, com parâmetros de entrada duas *strings*. A primeira representa o estado mais atual das vagas, o que acabou de ser recebido do Arduíno, e a segunda o estado passado. No método, é comparada a *string* anterior com a nova para checar o que é necessário ser executado. Caso a vaga tenha se mantido vazia, nada é feito. Caso a vaga tenha sido ocupada é necessário atualizar a informação do estado da vaga no banco, aumentar em 1 (um) o número de vezes que ela foi ocupada e aumentar o tempo que ocupação por 15 segundos, já que esse é o intervalo de tempo de checagem dos sensores. É feito também o aumento de 1 (um) no número de vagas ocupadas no setor em que a vaga se encontra. Caso a vaga tenha se mantido ocupada, é feito apenas o acréscimo de tempo ocupação da vaga no banco de dados. Se a vaga foi desocupada, atualiza-se seu estado no banco de dados para vazio e diminui-se o número de vagas ocupadas em seu setor.

2.2.2. Aplicação Servidora

Com todas as informações presentes no banco de dados e sendo atualizadas constantemente, é necessário haver um método para que os clientes possam usar estes dados. O responsável por isso é a aplicação servidora. As funções da aplicação servidora são gerenciar os pedidos dos clientes, requisitar do banco os dados necessários e enviar-lhes a resposta. Este servidor tem o maior número de métodos do sistema.

O primeiro passo é fazer que esse servidor possa ser acessado, para isso foi construído um **ServerSocket** na porta 6667, criando assim um endereço constante em que ele possa ser alcançado pelos clientes.

Para poder entender de forma correta os pedidos, foi criado um protocolo que deve ser seguido pelo servidor e pelo cliente. Neste protocolo o cliente deve enviar o pedido em uma *string*. Nela estarão as informações necessárias em uma espécie de código. O primeiro caractere da *string* se refere ao método que o cliente deseja invocar no servidor para receber os dados necessários, e os outros caracteres, separados por vírgula, se referem aos parâmetros deste método, caso eles existam. Por exemplo, caso o cliente queira invocar o método que devolve o mapa do estado de vagas no estacionamento, **MapaEst()**, ele deve mandar na *string* 0 (zero), que é o código referente ao método e também o mais um número, que é um parâmetro do método que se refere a identificação do estacionamento do qual ele deseja o mapa. No caso do sistema o estacionamento exemplo tem como código o 1 (um). Ou seja, o cliente tem que enviar uma *string* “0,1”.

O servidor deve, ao receber um pedido, tratá-lo para entender o que o cliente deseja. É utilizado um *split*, que quebra os caracteres da *string* entre as vírgulas. Um *split* é uma função em Java que quebra uma *string* em *strings* menores a partir de um caractere definido, nesse caso uma vírgula.

Para direcionar a execução da aplicação para o método correto utilizou-se *switch* e *case*. Nele, apenas certas partes do código são chamadas de acordo com o valor de um

parâmetro. Nesse caso o código referente ao método, ou seja, o cliente envia "0,1", o *case* 0 (zero) é ativado e é chamado o método 0 (zero) que é o **MapaEst()**. Funciona da mesma forma para todos os outros métodos, mudando apenas o valor do código.

O método que podem ser chamadas além do **MapaEst()** são o **VagaEst()**, que devolve o estado de uma vaga apenas, o **Pagar()**, que paga um tíquete, necessitando do seu código de barras. Pode ser chamado o **OcupSetor()**, que devolve informações de ocupação dos setores, necessitando da identificação do setor. Por fim existe o **Rel()**, que devolve um relatório que conta com estatísticas sobre o uso do estacionamento e tem como parâmetro a identificação do estacionamento. As informações devolvidas são quantas vezes uma vaga foi ocupada, por quanto tempo e uma média de tempo por ocupação.

Cada método requer uma consulta ao banco de dados, que devolverá vários dados. A partir das informações recebidas, o servidor cria um objeto referente ao pedido, como uma vaga ou um setor, e por *socket*, as retorna ao cliente. Para enviar um objeto por *socket* é necessário chamar um **ObjectOutputStream()**, que quebra esse objeto em uma cadeia de bits para que se possam ser transmitidos pela rede.É preciso que a classe deste objeto permita essa quebra, ela sinaliza a permissão implementando a interface **Serializable**.

Podendo agora enviar a resposta, o servidor cria um *socket*, endereçado para o IP (*Internet Protocol*) de origem de onde ele recebeu o pedido. A porta usada pelo cliente para receber as respostas é a 6668. Para que o cliente consiga traduzir de volta essa cadeia de *string* em objeto, ele precisa das mesmas classes presentes no servidor, que servirão com um modelo para a reconstrução.

2.3. Banco de Dados

O sistema de gerenciamento de bancos de dados utilizado foi o MySQL Server 5.6. Para manipulação e construção do modelo relacional foi empregado o MySQL Workbench 5.6. A Figura 3 exibe o modelo relacional do banco de dados:

A tabela **Estacionamento** possui com parâmetros **idEstacionamento**, que é a chave primária. Uma chave primária é única e ela é responsável pela identificação de

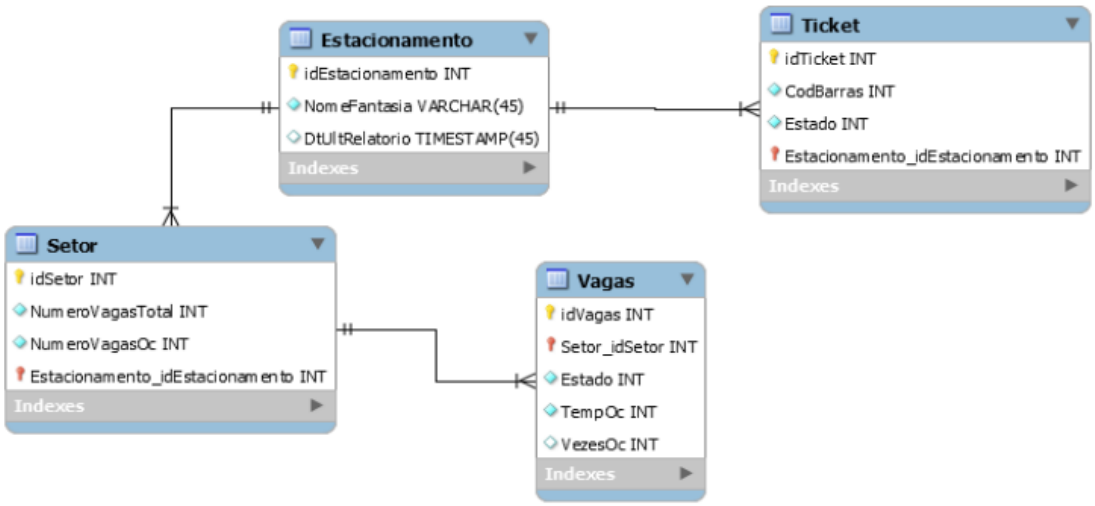


Figura 3. Modelo relacional do banco de dados

cada linha da tabela. A tabela também conta com **NomeFantasia**, que é um nome pelo qual o estacionamento é conhecido e também o **DtUltRelatorio**, que guarda a data do último relatório de estatísticas de ocupação emitido.

A tabela **Setor** possui **idSetor** como chave primária, **NumeroVagasTotal**, que guarda o total de vagas do setor e **NumeroVagasOc**, que guarda o número de vagas atualmente ocupadas. Na tabela também existe uma chave estrangeira **Estacionamento_idEstacionamento**, uma chave deste tipo é apresentada quando há um relacionamento entre tabelas, neste caso, um setor está relacionado a um estacionamento.

A tabela **Vagas** possui **idVaga** como chave primária, **Setor_idSsetor** como chave estrangeira que a relaciona com setor, **Estado**, que guarda o estado atual da vaga, com 0 (zero) para vazia e 1 (um) para ocupada, **TempoOc**, que guarda o tempo que aquela vaga ficou ocupada e **VezesOc**, que guarda quantas vezes aquela vaga foi ocupada.

A última tabela é a **Ticket**, que possui **idTicket** representando a chave primária, o **CodBarras**, que guarda o código de barras do tíquete, tem o Estado, que diz se o tíquete está pago (um) ou não (zero) e uma chave estrangeira, a **Estacionamento_idEstacionamento**, que o relaciona com Estacionamento.

2.4. Componente *Mobile*

Uma aplicação móvel cliente foi desenvolvida, integrada ao sistema base previamente descrito, garantindo mobilidade ao sistema e gerando novas funcionalidades. A aplicação conta com as seguintes funcionalidades: visualizar lista de estabelecimentos cadastrados no serviço, visualizar os estabelecimentos por perto, visualizar o mapa de vagas, geral e por setor, assim como o estado destas mesmas vagas, cadastro da vaga de um estacionamento para futuras notificações pertinentes a esta vaga e pagamento do tíquete do estacionamento. As Figuras 4, 5, 6 e 7 exibem os *mockups* das telas da aplicação construídos com a ajuda da ferramenta Balsamiq Mockups 3, exibindo um panorama geral de navegação da aplicação.

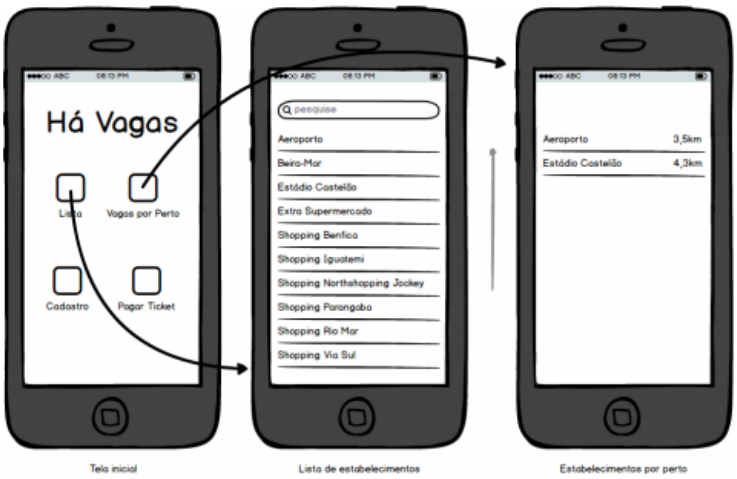


Figura 4. *Mockups* de tela: inicial, lista de estabelecimentos, e estabelecimentos por perto

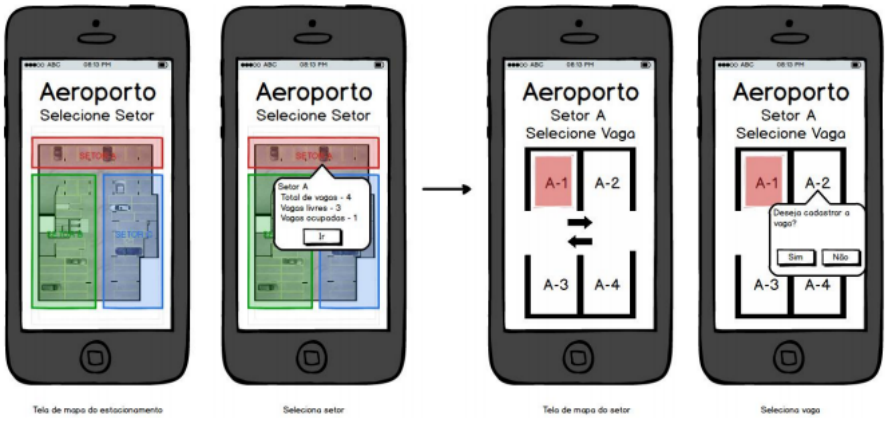


Figura 5. *Mockups* de tela: visualização de mapas e estado das vagas

Figura 6. *Mockups* de tela: cadastro e notificação

Figura 7. *Mockups* de tela: pagamento de tíquete

2.5. Possíveis Novos Componentes

O sistema de gerenciamento de vagas de estacionamento exposto neste trabalho (Figura 1) é um sistema que tem como uma de suas premissas a separação de seus componentes, conceito importante para superar desafios como a heterogeneidade das aplicações finais. Desta maneira, diferentes aplicações finais podem ser integradas ao sistema e consigo mais recursos e funcionalidades.

São inúmeras as funcionalidades que podem ser implementadas em um serviço de estacionamento. Baseado no modelo de negócios conhecido de gerências de estacionamentos, o sistema poderia, por exemplo, contar com uma página *web* ou ser posto como serviço de modo integrado aos próprios websites dos estabelecimentos já em funcionamento, onde seriam ofertadas funcionalidades como exibição do mapa de vagas, pagamento de tíquete, informações sobre preços e promoções, publicidades relacionadas ao estabelecimento, entre outras. Esta ferramenta poderia ser integrada ao sistema na forma de *web services*.

Outro componente que poderia ser integrado ao sistema seria um totem provido de uma tela *touchscreen* localizada idealmente nas cancelas de entradas e saídas dos estacionamentos, onde os usuários poderiam rapidamente usufruir do recurso de visualização do mapa das vagas, colhendo informações como nível de ocupação por setor, número de vagas livres e ocupadas por setor, localização de vagas cobertas, localização de vagas especiais para pessoas deficientes, entre outros. Desse modo, facilitando a vida do usuário na escolha e consulta das vagas do estacionamento.

3. Estratégias de Desenvolvimento da Aplicação Móvel

O desenvolvimento de aplicativos móveis cresce exponencialmente em todo o mundo. São inúmeras as possibilidades de desenvolvimento de aplicações moveis, podendo serem na forma de aplicações *web*, aplicativos nativos e ainda híbridos. Entretanto, a escolha do tipo de aplicativo a ser construído requer uma análise minuciosa de vários fatores como: plataforma de desenvolvimento, sistema a ser construído, serviços a serem implementados e arquitetura [Luquetti et al. 2015].

Neste trabalho optou-se pela construção de uma aplicação nativa na plataforma Android. O uso de recursos como GPS e câmera na plataforma de desenvolvimento podem ser melhor explorados dentro das funcionalidades da aplicação. Outro aspecto foi o fato da praticidade no desenvolvimento, já que a plataforma utiliza da linguagem Java, linguagem imensamente estudada no âmbito acadêmico.

3.1. Características da Aplicação

3.1.1. Requisitos

A aplicação foi desenvolvida para executar em dispositivos móveis que usem o sistema operacional Android versão 2.2 ou superior, utilizando recursos do dispositivo como GPS e câmera. Suas funcionalidades são: visualizar lista de estabelecimentos cadastrados no serviço, visualizar os estabelecimentos por perto, visualizar o mapa de vagas, geral e por setor, assim como o estado destas mesmas vagas, cadastro da vaga de um estacionamento para futuras notificações pertinentes a esta vaga e pagamento do tíquete do estacionamento.

Na tela principal são exibidas as seguintes funcionalidades: listar, vagas por perto, cadastrar e pagar tíquete. Ao selecionar a opção "listar", listam-se todos os estabelecimentos que possuem estacionamento que estão cadastrados no banco de dados do sistema. Nesta tela o usuário poderá selecionar um estabelecimento, sendo direcionado a uma tela onde será exibido o mapa do estabelecimento e os setores deste estacionamento. Ao selecionar um setor qualquer serão exibidas informações referentes ao setor como número de vagas, números de vagas vazias e número de vagas ocupadas, com a opção de seguir para o mapa do setor selecionado ou não. Ao seguir para a visualização do mapa do setor, será exibido o mapa do setor propriamente dito contendo todas as vagas do setor, podendo estar nos estados vazia ou ocupada. Ao se clicar em uma vaga que se deseja cadastrar e se essa vaga já não for cadastrada no sistema, o usuário será direcionado para a tela de cadastro de vaga.

Ao selecionar a opção "vagas por perto", o usuário será direcionado a uma tela de mapa mediante uso da API de mapas do Google com marcadores que representam a sua localização atual e os estabelecimentos cadastrados no sistema. Ao clicar em um marcador de estabelecimento no mapa, a aplicação exibirá uma caixa de mensagem reportando o nome do estabelecimento, a distância entre ele e o usuário e a opção de seguir para o estabelecimento. Ao seguir para o estabelecimento o usuário poderá interagir com a aplicação da mesma maneira que é descrito anteriormente quando se seleciona um estabelecimento da lista na opção "listar".

Ao selecionar a opção "cadastrar", o usuário será direcionado para uma tela de cadastro onde poderá cadastrar de uma vaga qualquer, informando o nome da vaga e um número de telefone. Após o cadastro, os dados serão armazenados para eventuais notificações sobre a vaga referida.

Por fim, ao selecionar a opção "pagar tíquete", a aplicação exibirá um leitor de código de barras e solicitará uma leitura. Caso a aplicação leia um código cadastro no sistema, será exibida uma tela com o número referente ao código lido e o valor daquele código. Será realizada uma simulação de pagamento e exibido um retorno de pagamento realizado com sucesso. Caso o código de barras não esteja cadastrado no sistema será exibida uma mensagem indicando o não cadastramento do mesmo.

3.1.2. Ambiente de Desenvolvimento

Para o desenvolvimento de aplicativos em Android é necessário que esteja configurado um ambiente contendo a última versão do *Java Development Kit* (JDK) juntamente com o Android SDK.

Para o desenvolvimento da aplicação móvel utilizou-se IDE (*Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado) Eclipse na sua versão 4.5.2 (codinome Mars.2), pois é uma IDE de código aberto e com a opção de extensão de suas funcionalidades por meio da instalação de vários *plugins* disponíveis, dentre eles o ADT, que é um *plugin* desenvolvido para estender as funcionalidades do Eclipse possibilitando a criação de projetos voltados para o Android. Este *plugin* utiliza as bibliotecas disponibilizadas pelo SDK, funcionando como uma ponte que une o Android SDK ao Eclipse.

Estando instalado e configurado o ambiente (IDE + ADT + SDK), é necessário a criação de um *Advanced Virtual Device* (AVD), para que se possa executar e testar a aplicação desenvolvida. O AVD é um dispositivo virtual do sistema operacional do Android que simula um aparelho no qual se pode executar e testar aplicações. Pode-se criar vários AVDs de acordo com o tipo de aparelho que se deseje emular e versão da plataforma Android na qual o projeto está sendo desenvolvendo. Um AVD é basicamente composto de um perfil de hardware (possui câmera, utiliza teclado, etc), uma versão da plataforma Android a executar, um emulador de cartão SD e uma área de armazenamento na máquina de desenvolvimento.

3.1.3. Diagramas

Um caso de uso representa a interação dos atores com o sistema e pode ser representado por uma descrição simples e em linguagem natural, ajudando a identificar os objetos e a compreender o que o sistema deve realizar [Sommerville 2003]. Na arquitetura são propostos seis casos de uso que representam funcionalidades disponíveis e interações dos atores envolvidos com o sistema. A Figura 8 exibe o diagrama de caso de uso referente à interação dos atores com a aplicação executada no dispositivo móvel. A Figura 9 exibe um diagrama de sequência onde é possível verificar o fluxo de interações do aplicativo.

3.2. Funcionalidades da Aplicação

A aplicação móvel conta com quatro funcionalidades básicas. Buscou-se retratar essas funcionalidades da melhor maneira possível no sentido de representação de imagens e usabilidade durante a construção das telas da aplicação. A Figura 10 (a) exibe a tela inicial com as funcionalidades implementadas na aplicação.

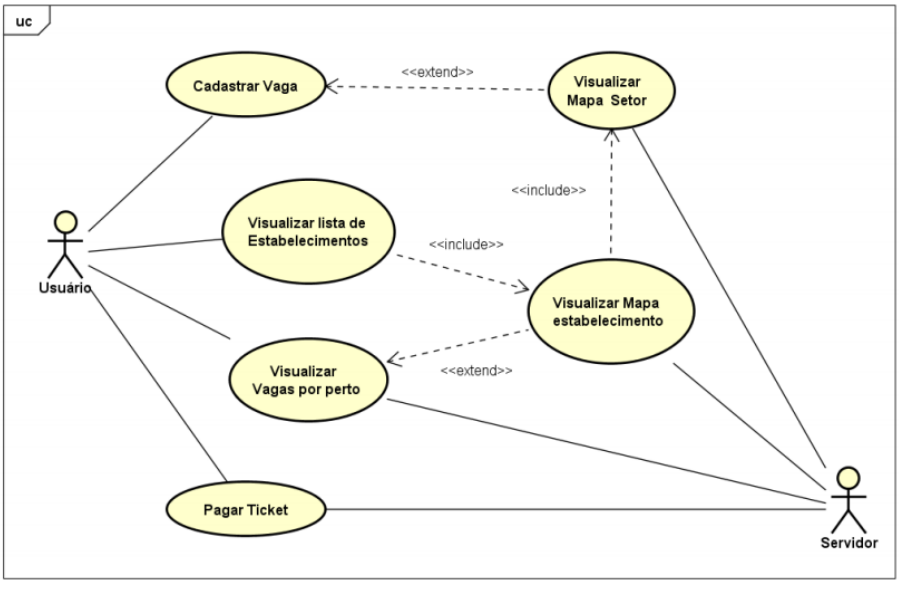


Figura 8. Caso de uso da aplicação móvel

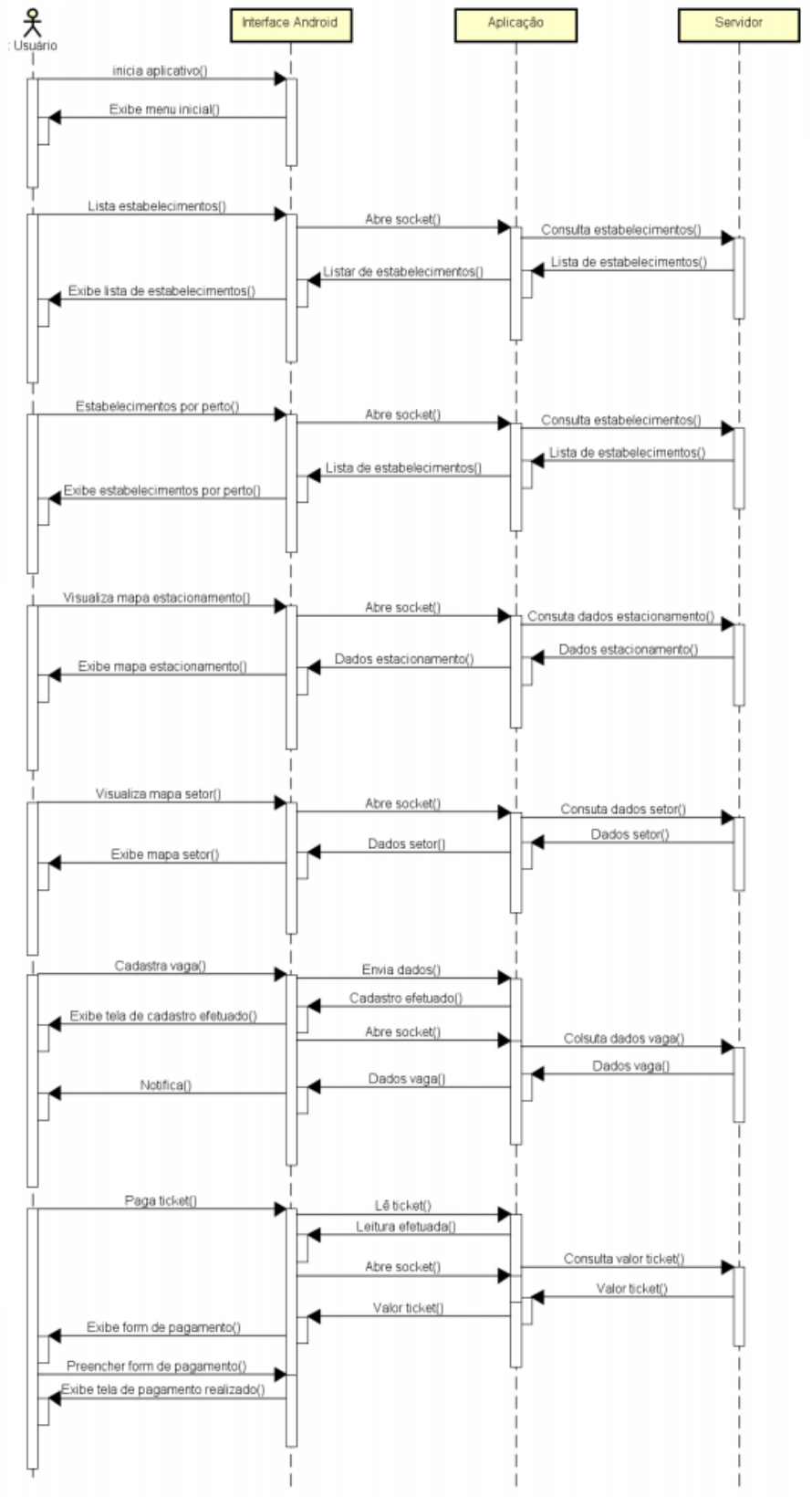
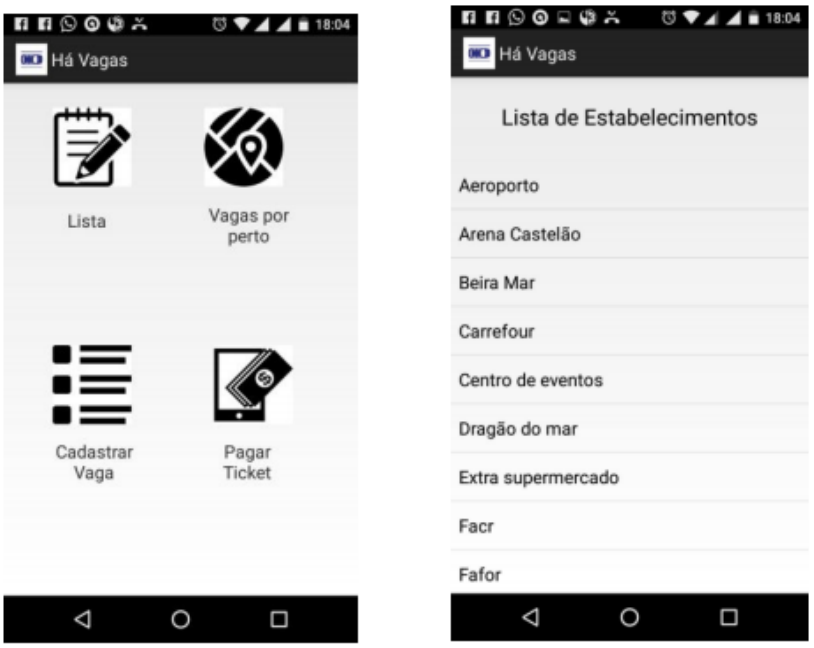


Figura 9. Diagrama de sequência da aplicação móvel



(a) Tela inicial da aplicação

(b) Tela da lista de estabelecimentos

Figura 10. Tela de acesso inicial da aplicação e lista de estabelecimentos

3.2.1. Visualizar Estabelecimentos

A primeira funcionalidade da aplicação é a de listar todos os estabelecimentos cadastrados no sistema. A Figura 10 (b) exibe a tela formada pelo resultado da coleta.

A outra maneira de visualizar os estabelecimentos cadastrados no sistema é pela funcionalidade "Vagas por Perto". Nela a aplicação utiliza da API *playservices* da Google para acessar a biblioteca referente à função **maps** e com ela exibir na tela o mapa do mundo. Por meio da API também são postos os marcadores que representam os estabelecimentos por meio de suas coordenadas (latitude e longitude).

Para utilização da API é necessário a importação das bibliotecas e vinculação ao projeto da aplicação, assim como criação de uma chave na *Google Play Services* que indicará que a aplicação estará autorizada para utilizar dos serviços da Google. Além disso, algumas permissões de acesso precisam ser descritas.

A chamada da localização atual do usuário e a inserção a da localização no mapa ocorrem por meio de métodos que obtém a posição atual do usuário. A Figura 11 exibe a tela do mapa com os componentes dispostos.

Ao selecionar o marcador que representa um estabelecimento cadastrado no sistema, será exibida a opção de seguir para o estabelecimento clicando na janela do marcador, e a aplicação exibe o mapa dividido por setores do estabelecimento.

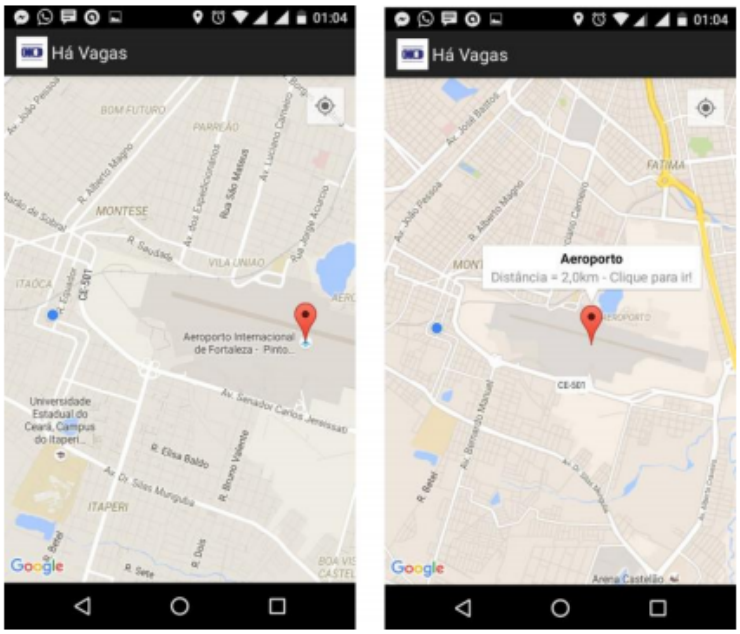
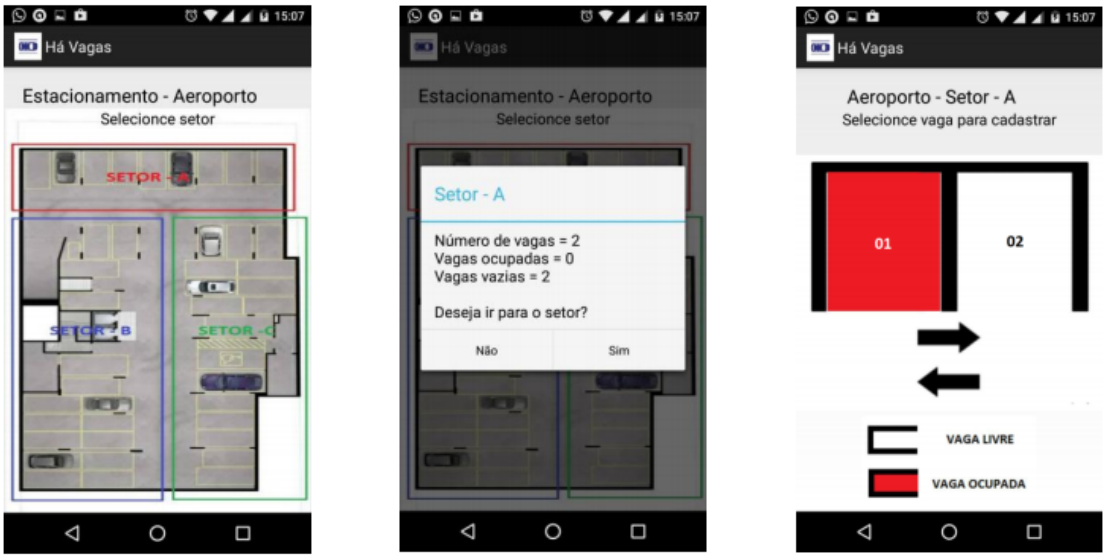


Figura 11. Tela referente ao mapa com marcadores referentes aos estabelecimentos

3.2.2. Visualizar Vagas

Em ambas as formas de apresentação dos estabelecimentos cadastrados (lista na tela de listagem dos estabelecimentos ou disposição dos estabelecimentos no GoogleMaps) os estabelecimentos podem ser selecionados para então direcionamento do usuário a uma tela referente ao mapa do estacionamento do estabelecimento selecionado. Nessa tela é exibido um mapa de estacionamento modelo representando o estacionamento do local selecionado, dividido por setores e com uma solicitação de escolha de setor. O mapa desenhado para validação deste protótipo foi meramente figurativo, com a única finalidade de proporcionar usabilidade às funcionalidades da aplicação, não sendo, assim, coletado do banco de dados do sistema. A Figura 12 (a) mostra a tela que exibe o mapa do estacionamento do estabelecimento selecionado.

Uma caixa de diálogo informará ao usuário o número de vagas que o setor possui, o número de vagas vazias, o número de vagas ocupadas e a pergunta se o usuário deseja ir para o setor selecionado. A Figura 12 (b) mostra a tela do mapa do estacionamento com o setor selecionado. Selecionando a opção “Não” na caixa de diálogo o usuário permanecerá na tela de mapa do estacionamento. Selecionando a opção “Sim”, o usuário se deparará com uma tela referente ao mapa do setor selecionado no estacionamento. Neste mapa serão exibidas todas as vagas do setor. Estas vagas são representadas pelos sensores que estão dispostos no componente embarcado do sistema, assim sendo, um único setor com duas vagas foi cadastrado no sistema, pois o sistema proposto neste trabalho dispõe de somente dois sensores. A Figura 12 (c) exibe a tela referente ao mapa do setor selecionado. A Figura 13 exibe a tela referente a uma vaga selecionada no mapa do setor.



(a) Tela referente ao mapa do estabelecimento

(b) Tela referente ao mapa do estacionamento com o setor selecionado

(c) Tela referente ao mapa do setor selecionado

Figura 12. Telas do estabelecimento

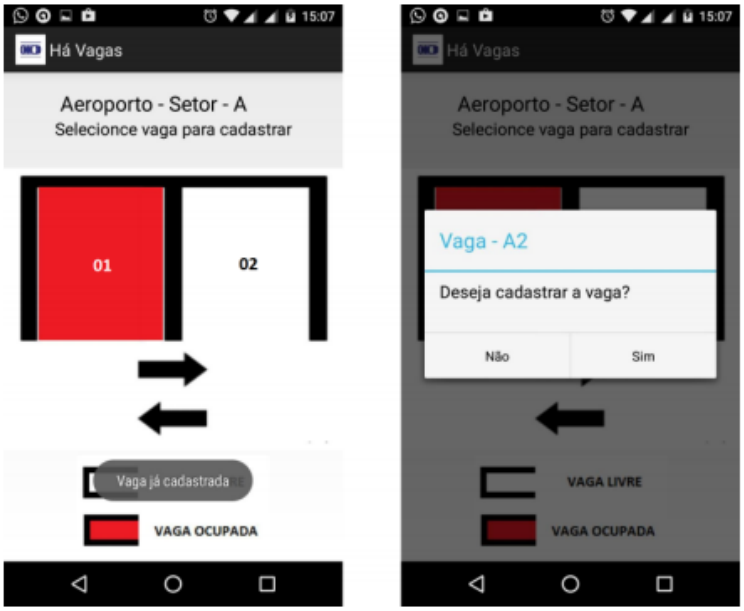


Figura 13. Tela referente a vaga selecionada no mapa do setor

3.2.3. Cadastro e Notificação

Após selecionar a opção "sim" na caixa de diálogo referente à vaga selecionada, ou ainda, selecionando o botão "Cadastrar Vaga" na tela inicial da aplicação, a aplicação é direcionada para um cadastro que representa uma tela de cadastro de vaga. Nela será solicitado ao usuário o nome da vaga (A2, por exemplo) e um número de telefone para constar como registro no banco de dados. A Figura 14 exibe a tela de cadastro de vaga. Caso haja

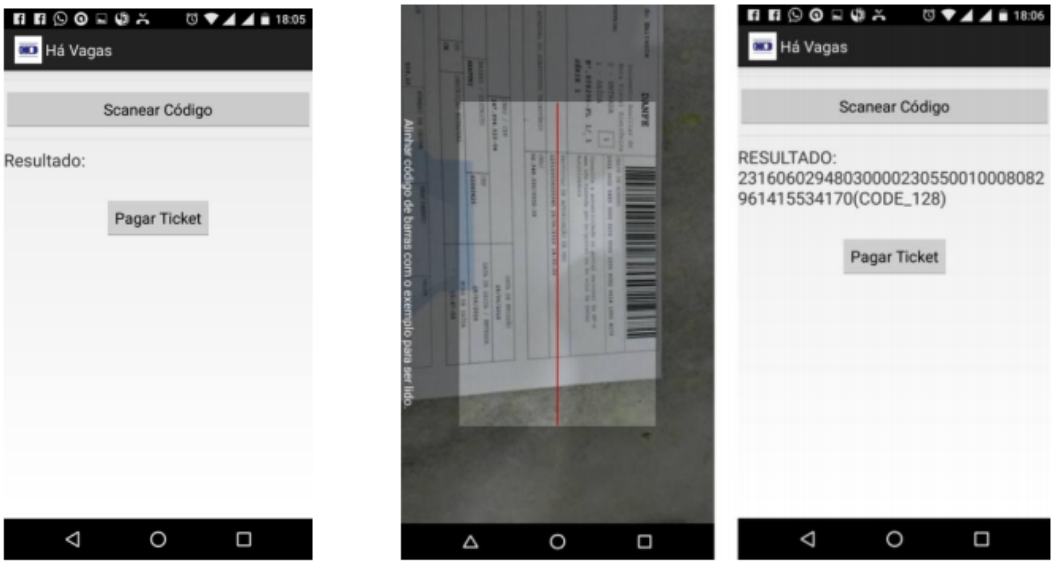
Figura 14. Tela referente ao cadastro de vaga

Figura 15. Tela com notificação recebida

alteração no estado da vaga de ocupado para vazio, a aplicação emitirá uma notificação ao usuário alertando-o da mudança de estado da sua vaga (Figura 15).

3.2.4. Pagamento

A função “Pagar tíquete” na tela inicial direciona usuário para uma tela para a leitura de um código de barras ou QR *code*, o código de barras lido para exibição tela e um botão para o pagamento do valor contido no código de barras. A Figura 16 (a) exibe a tela de pagamento. Ao se pressionar o botão para ler o código de barras, a aplicação irá abrir uma tela para a leitura pela câmera do dispositivo, ao capturar um código de barras qualquer a aplicação retornará para a tela de pagamento agora exibindo o resultado da leitura como mostra a Figura 16 (b).



(a) Tela de pagamento (b) Tela de pagamento após leitura de código

Figura 16. Telas de pagamento e leitura do código de barras

Ao ser realizada a leitura do código de barras e o resultado impresso em tela, o usuário então poderá pressionar o botão de pagamento de tíquete. A aplicação por sua vez exibirá uma tela de atividade contendo um simples formulário que simula o pagamento do tíquete de estacionamento lido. Ao preencher o formulário e pressionar o botão de pagamento, a aplicação exibirá uma mensagem de “pagamento realizado com sucesso” e retornará para a tela inicial. A Figura 17 exibe a tela referente ao formulário de pagamento.

4. Conclusão

São inúmeros os desafios relacionados ao emergente paradigma de computação ubíqua. Por si só, o desenvolvimento de sistemas dessa natureza não é tarefa trivial. Esses sistemas contam com recursos computacionais postos de formas invisíveis ao usuário e ao mesmo tempo proporcionam mobilidade a seus serviços. Diante dessas motivações, o trabalho apresenta de maneira prática formas de construção de sistemas e estratégias de integração entre partes do sistema, mas especificamente um componente móvel a um sistema base proposto.

O trabalho consistiu em desenvolver uma aplicação móvel e como esboçar estratégias de integração desta aplicação com um sistema distribuído. Tal sistema conta

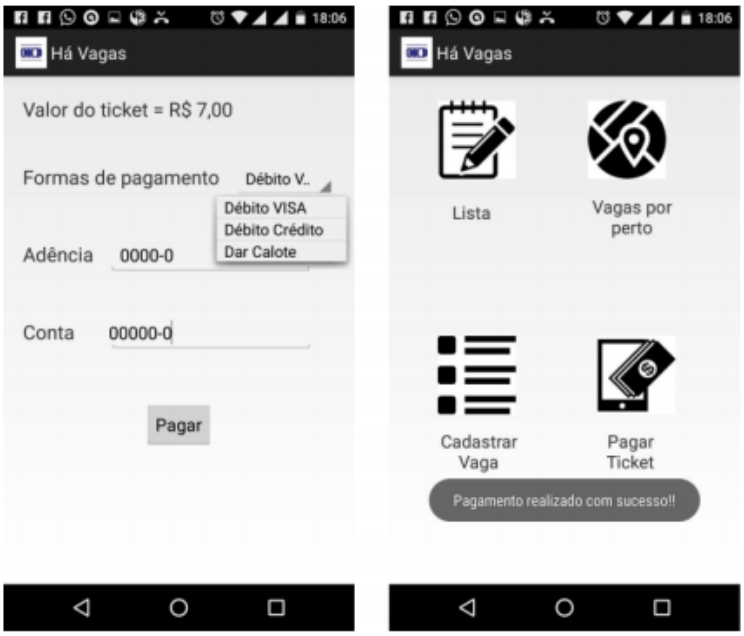


Figura 17. Tela de formulário para pagamento de tíquete

com componentes embarcados, servidores e banco de dados. Este sistema distribuído tem como função básica gerenciar as vagas de um estacionamento, possuindo uma rede de sensores para determinar o estado das vagas, comandadas por um microcontrolador Arduino, enviando informações coletadas a um servidor que as salva em um banco de dados, sendo gerenciados por outro servidor para que clientes possam utilizar estas informações em suas funcionalidades.

A escolha da plataforma Android para o desenvolvimento da aplicação foi motivada principalmente por ser um projeto de código aberto com disponibilidade de vasta documentação e códigos de exemplo, além de ferramentas gratuitas que auxiliam o desenvolvimento.

No âmbito acadêmico do curso de Bacharelado em Sistema e Mídias Digitais, da Universidade Federal do Ceará, o sistema desenvolvido apresenta inúmeros conceitos estudados durante todo o curso. Disciplinas como Lógica de Programação, Programação Orientada a Objetos e Estruturas de Dados foram responsáveis pelo conhecimento de conceitos básicos de programação e linguagem Java. Redes de Computadores e Sistemas Distribuídos ensinaram conceitos sobre utilizados massivamente na construção do sistema. Introdução a Banco de Dados e Bancos de Dados Aplicados a Multimídia ajudaram na construção do banco de dados da aplicação. Engenharia de Software ajudou na questão da construção do escopo do projeto do sistema. Sistemas Embarcados foi de fundamental importância no desenvolvimento da rede de sensores presentes no sistema. Por fim, a disciplina de Introdução à Computação Móvel e Ubíqua contribuiu imensamente no desenvolvimento da aplicação móvel.

Um problema encontrado durante o desenvolvimento do projeto foi o acesso aos materiais aos componentes embarcados, uma vez que a venda de sensores, placa Arduino e outros componentes básicos não são fáceis de encontrar localmente. Outra dificuldade

encontrada se dá ao fato de não haver meios de execução de testes nas aplicações clientes no que tange a parte de integração destas com o sistema base, o que torna o ciclo de desenvolvimento bastante procedural.

Tendo como resultado a construção de um sistema distribuído que gerencia vagas de estacionamento, a documentação e especificações da construção do sistema assim como o próprio sistema em si poderiam ser usadas como base para o desenvolvimento e incorporação de novos componentes ao sistema, referentes ao contexto do problema que é a gerencia de estacionamentos. Estes componentes podem abordar outros aspectos do desenvolvimento de sistemas, como um componente *web*, que abordariam tecnologias como *webservices* ou aspectos como comunicação remota.

Novas funcionalidades também poderiam ser implementadas à aplicação móvel como: definir rota para a melhor vaga; indicar previamente a vaga vazia mais próxima das entradas dos estabelecimentos; integração com redes sociais, entre outras. Por se tratar de um sistema distribuído ubíquo, o sistema construído neste trabalho poderia servir como base para o estudo e desenvolvimento de estratégias que visem os desafios relacionados a este paradigma de computação, como o aspecto da escalabilidade do sistema, do tratamento da concorrência da informação, da segurança dos dados da heterogeneidade do sistema e ainda da capacidade de adequação ao contexto.

Referências

Barros, E. e Cavalcante, S. (2016). Introdução aos sistemas embarcados. http://www.cin.ufpe.br/∼vba/periodos/8th/s.e/aulas/STP%20-%20Intro%20Sist%20Embarcados.pdf. Online; acessado em maio-2016.

Coulouris, G., Dollimore, J., e Kindberg, T. (2013). *Sistemas Distribuídos - Conceitos e Projeto*. Bookman, 5 edition.

de Araujo, R. B. (2003). *Computação Ubíqua: Princípios, Tecnologias e Desafios*. SBC - XXI Simposio Brasileiro de Redes de Computadores (SBRC) - Natal.

Luquetti, L., Facciolo, D., e Carvalho, S. (2015). Projetos do artigo “desenvolvimento de aplicações para dispositivos móveis: Tipos e exemplo de aplicação na plataforma ios”. https://github.com/leandroluquetti. Online; acessado em maio-2016.

Rxtx (2016). Rxtx. http://rxtx.qbang.org/wiki/index.php/Main_Page. Online; acessado em maio-2016.

Sommerville, I. (2003). *Engenharia de Software*. Pearson, 6 edition.