INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SUL-RIO-GRANDENSE - CÂMPUS PASSO FUNDO
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

VAINER JOEL GROTH

ANÁLISE E DESENVOLVIMENTO DE UM SISTEMA CRM PARA CONCESSIONÁRIAS DE VEÍCULOS

Jorge Luis Boeira Bavaresco

PASSO FUNDO
2015

VAINER JOEL GROTH

ANÁLISE E DESENVOLVIMENTO DE UM SISTEMA CRM PARA CONCESSIONÁRIAS DE VEÍCULOS

Monografia apresentada ao Curso de Tecnologia em Sistemas para Internet do Instituto Federal Sul-Rio-Grandense, Campus Passo Fundo, como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet.

Orientador: Jorge Luis Boeira Bavaresco

PASSO FUNDO

2015

VAINER JOEL GROTH

ANÁLISE E DESENVOLVIMENTO DE UM SISTEMA CRM PARA CONCESSIONÁRIAS DE VEÍCULOS

Trabalho de Conclusão de Curso aprovado em ____/____/____ como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet

Banca Examinadora:

_____

Prof. Jorge Luis Boeira Bavaresco

Orientador

_____

Prof. Alexandre Tagliari Lazzaretti

Avaliador

_____

Prof.ª Carmen Vera Scorsatto

Avaliadora

_____

Prof. Adilso Nunes de Souza

Coordenador do Curso

PASSO FUNDO

2015

DEDICATÓRIA

À Deus,
por me manter firme na busca da realização de um sonho.
Aos meus pais,
pelo amor, apoio e incentivo a seguir na vida acadêmica.
À minha esposa,
pela compreensão e apoio nas decisões que tomei.
Aos professores,
pelos ensinamentos, conselhos e experiências compartilhadas.

“*Ninguém ignora tudo.*

Ninguém sabe tudo.

Todos nós sabemos alguma coisa.

Todos nós ignoramos alguma coisa.

Por isso aprendemos sempre.”

Paulo Freire

RESUMO

No mercado atual, empresas que realizam uma boa gestão do relacionamento com os seus clientes, ganham preciosos pontos para com a concorrência, agilizando a tomada de decisões, com o intuito de garantir a satisfação de seus clientes. Este projeto destina-se a análise e desenvolvimento de um software CRM para concessionárias de veículos, tomando por base um estudo de caso, realizado em uma concessionária atuante na região de Passo Fundo. O estudo de caso foi baseado na análise do processo atual, de controle e gerenciamento de contatos realizados entre clientes e vendedores da concessionária, com o objetivo de desenvolver um software, a fim de implementar melhorias ao processo. Para desenvolvimento do software foram utilizadas as tecnologias JavaFX, JPA com framework Hibernate e banco de dados PostgreSQL, ao final da pesquisa o objetivo principal foi atingido, utilizando-se destas tecnologias no desenvolvimento de um software CRM para uma concessionária de veículos.

Palavras Chave: JavaFX; interfaces ricas; software CRM.

ABSTRACT

In today's market, companies that perform a good management of the relationship with their clients, earn precious points for the competition, accelerating the decision-making process, in order to ensure the satisfaction of its customers. This project is intended for the analysis and development of a CRM software for car dealerships, based on a case study, carried out in a specialized in the area of Passo Fundo. The case study was based on analysis of the current process, control and management of contacts made between customers and vendors of the concessionaire, with the goal of developing a software, in order to implement improvements to the process. For software development were used the JavaFX technologies, JPA with framework Hibernate and PostgreSQL database, at the end of the survey the main objective was reached, using these technologies in the development of a CRM software for a vehicle dealership.

Keywords: JavaFX; rich interfaces; CRM software.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

RIA: *Rich Internet Application*, p. 14
CRM: *Customer Relationship Management*, p. 14
JPA: *Java Persistence API*, p. 16
ERP: *Enterprise Resource Planning*, p. 17
GUI: *Graphical User Interface*, p. 19
AJAX: *Asynchronous Java Script and XML*, p. 18
HTML: *Hyper Text Markup Language*, p. 18
GPU: *Graphics Processing Unit*, p. 20
F3: *Form Follows Function*, p. 20
API: *Aplication Programming Interface*, p. 21
XML: *Extensible Markup Language*, p. 23
MVC: *Model-View-Controller,* p. 24
IDE: *Integrated Development Environment*, p. 25
UML: *Unified Modeling Language*, p. 27
SGBD: *Sistema de Gerenciamento de Banco de Dados*, p. 27
DMS: *Dealer Management System*, p. 28
SUV: *Sport Utility Vehicle*, p. 28
QCP: *Qualidade Compromisso e Participação*, p. 34
CSS: *Cascading Style Sheets*, p. 56

SUMÁRIO

1 INTRODUÇÃO

Sistemas baseados na web estão cada vez mais presentes, seja no desenvolvimento de novas aplicações ou substituindo aplicações desktop que, de certa forma se tornaram obsoletas. São inúmeras as vantagens encontradas em um sistema web, geralmente em ambientes corporativos, podem não oferecer a mesma usabilidade de um sistema desktop (AGOSTINI e RODRIGUES, 2010).

Aplicações RIA (Rich Internet Application) surgiram com o intuito de aproximar os dois mundos, web e desktop, utilizando da mobilidade encontrada nos sistemas web com a usabilidade e componentes oferecidos por sistemas desktop (AGOSTINI e RODRIGUES, 2010). JavaFX vem crescendo constantemente no mundo das aplicações RIA, recentemente deixou de ser apenas uma API separada para contemplar nativamente o Java 8 (DEA et al, 2014).

No presente trabalho é apresentado um estudo sobre a tecnologia JavaFX no desenvolvimento de um sistema CRM (Customer Relationship Management) para uma concessionária de veículos. É apresentada uma modelagem do sistema baseado em um estudo de caso, tentando suprir as necessidades e amenizar as dificuldades encontradas no processo atual.

O Objetivo é utilizar os recursos oferecidos pela tecnologia JavaFX para o desenvolvimento futuro de um sistema CRM capaz de agilizar a tomada de decisões e auxiliar a empresa em seus objetivos organizacionais.

A motivação para desenvolvimento deste projeto é a possibilidade de aprender sobre a tecnologia JavaFX e aplicar os seus recursos a fim de implementar melhorias no processo de atendimento a clientes do setor de vendas de uma concessionária de veículos. O sistema não foi plenamente finalizado ao término deste projeto, mas é possível realizar a manutenção de cidades e pessoas, bem como revendas e usuários, este ainda necessita de ajustes relativos a permissões. No módulo CRM é possível realizar a manutenção de locais de contatos, motivos de venda perdida e modelos e acessórios de veículos, por fim permite a manutenção de atendimentos, bem como os contatos realizados.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Desenvolver um sistema CRM para uma concessionária de veículos, a fim de aperfeiçoar o atendimento prestado por sua equipe de vendas, além de manter um histórico dos atendimentos por eles realizados.

1.1.2 Objetivos Específicos

O trabalho tem como base os seguintes objetivos específicos:

- Estudar a tecnologia JavaFX e suas principais características;
- Estudar o mapeamento objeto-relacional através de JPA com framework Hibernate e Banco de Dados PostgreSQL;
- Analisar o sistema utilizado para controle dos atendimentos realizados por uma concessionária de veículos;
- Modelar um sistema de CRM com base no estudo de caso proposto;
- Programar um sistema CRM para controle dos atendimentos realizados pelos vendedores de uma concessionária de veículos, utilizando as tecnologias acima citadas.

2 REFERENCIAL TEÓRICO

Esta seção visa dar embasamento teórico ao projeto desenvolvido, aborda conceitos sobre CRM, aplicações de interface RIA, JPA (Java Persistence API) e conceitos sobre a tecnologia JavaFX. Veremos como a utilização destas tecnologias pode auxiliar no desenvolvimento da aplicação.

2.1 CRM

O termo CRM é utilizado para definição do gerenciamento com foco no cliente, ou seja, colocando-o na frente de outras situações, tentando prever e suprir as suas necessidades. Em um âmbito geral, abrange a maioria das áreas da empresa, estando presente efetivamente em áreas como marketing, vendas e serviços.

É comum encontrarmos diversas definições para o termo CRM, segundo Marshak apud (GREENBERG, 2001, p.5) "há tantas definições de CRM quantas são as empresas vendendo soluções para se aproximar do cliente". Já Peter Keen, presidente da Keen Innovations, apud (GREENBERG, 2001, p.5) lança sua definição ideal de CRM:

> CRM é o comprometimento da empresa em colocar a experiência do cliente no centro de suas prioridades e em garantir que sistemas de incentivo, processos e fontes de informação alavanquem o relacionamento por meio da melhoria da experiência, conquistando mais confiança e um senso de valor pessoal por parte dos clientes.

Entre diversas formas de se trabalhar com CRM, a mais ágil e eficaz é através da implantação de um software CRM, disposto de ferramentas para auxiliar a rotina dos colaboradores e os contatos realizados com clientes da empresa (GREENBERG, 2001).

Apesar da definição do termo CRM ser único, independente do ramo da empresa, é possível encontrar softwares de CRM específicos para determinadas áreas, por exemplo, um software considerado ideal para controle de uma loja de venda de aparelhos celulares pode não ser ideal para o controle de um telemarketing, ou então uma oficina mecânica, mas ambas podem obter excelentes

resultados fazendo uso de um software CRM mais adaptado as suas necessidades (GREENBERG, 2001).

Segundo Greenberg (2001, p.11) “Um bom portal de CRM agrega toda a informação referente ao consumidor em um único aplicativo ou desktop, em formato customizado para quem utiliza os dados, visando à melhoria da interação entre a empresa e o cliente”.

Há situações em que empresas já utilizam seus softwares para controle de suas atividades, comumente denominados de ERP (Enterprise Resource Planning), estes por sua vez, geralmente são voltados para o controle interno, executando emissão de Notas Fiscais, controle de Ordens de Serviço, mantendo banco de dados de clientes e fornecedores, enfim, auxiliando nas tarefas diárias, mas o foco principal não é o relacionamento com o cliente, mas sim voltado para o interior da empresa (GREENBERG, 2001).

Alguns sistemas de ERP possuem um módulo de CRM, mas este por sua vez, pode não atender as necessidades da empresa, surge então à necessidade da implantação de um Software CRM paralelo ao Sistema de ERP. Apesar de cada um executar tarefas distintas podem interagir entre si, como por exemplo, compartilhar o mesmo Banco de Dados, uma tarefa que pode não ser tão simples assim, já que estas arquiteturas distintas podem nem sempre ser compatíveis, como citou Greenberg:

> Muito dinheiro foi gasto com aquisição e implementação de sistema, mas de ERP, mas talvez maiores tenham sido os problemas gerados pela onda. Por volta de 1999, esses sistemas, totalmente concentrados no interior das empresas e em seus processos, perderam muito espaço. Na mesma época, as soluções de CRM, voltadas ao cliente, ao relacionamento e, portanto, às vendas, e não à produção e ao controle, estavam em cena. A integração entre ambos passou a ser o ponto crucial: back office em uma só sintonia. Fácil de falar, dificílimo de implementar, já que as arquiteturas e lógicas de desenvolvimento dos dois tipos de soluções eram incompatíveis! (GREENBERG, 2001, p.28)

O simples fato de implantar um sistema CRM não significa que a empresa irá ficar totalmente voltada para o cliente, mas se deseja acompanhar o ciclo de vida deste perante a empresa, é imprescindível o auxílio de um sistema CRM, a ideia é tentar antecipar-se a concorrência no suprimento das necessidades ou desejos de consumo de seus clientes (GREENBERG, 2001).

2.2 RIA

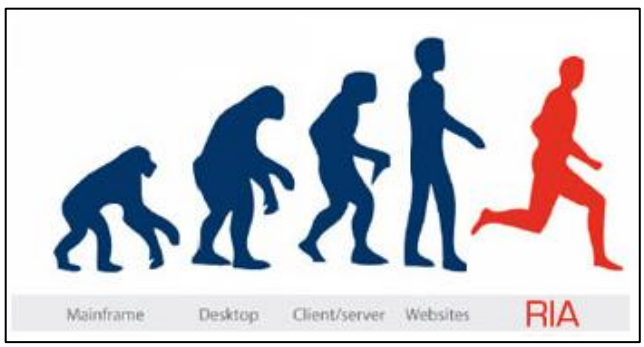
Aplicações RIA traduzidas para o português: Aplicações Ricas para Internet, surgiram com a ideia de aproximar aplicações web de aplicações desktop, com o intuito de levar às aplicações web, caracterizadas por sua mobilidade, a mesma usabilidade encontrada nas aplicações desktop (AGOSTINI e RODRIGUES, 2010). Apesar do termo RIA caracterizar aplicações web, seu conceito tem uma abrangência mais ampla, busca simplificar e padronizar interfaces de usuário, independente de navegador, plataforma ou dispositivo (PREMKUMAR apud MARQUEZINI, 2013).

As primeiras aplicações a carregar o conceito RIA surgiram em 1995 com o Java Runtime, especificamente através da criação das applets, que permitiam aos desenvolvedores criar aplicações com maior complexidade e riqueza de detalhes.

As applets surgiram para suprir a falta de recursos do Java Script e HTML da época, além disso, as applets eram multiplataforma, capaz de serem executadas em diferentes navegadores e plataformas (PINA e OLIVEIRA, 2013).

Com o passar do tempo tanto Java Script quanto HTML evoluíram e supriram carências antes encontradas. Com o surgimento do AJAX (Asynchronous Java Script and XML), que criou métodos de desenvolvimento focados especificamente em Java Script e HTML, que possibilitou a criação de interfaces mais ricas, aprimorando a usabilidade do usuário (PINA e OLIVEIRA, 2013). A Figura 1: Evolução das interfaces de usuário a seguir faz uma analogia à evolução da tecnologia para criação de interfaces.

Figura 1: Evolução das interfaces de usuário



Fonte: Desenvolvedores.net[1]

Com a evolução das tecnologias RIA, surgiram diversas plataformas oferecendo aos desenvolvedores, técnicas diferenciadas para criação de interfaces, as mais comuns usadas atualmente são: Adobe Flash, Microsoft Silverlight, HTML5 e JavaFX (PINA e OLIVEIRA, 2013).

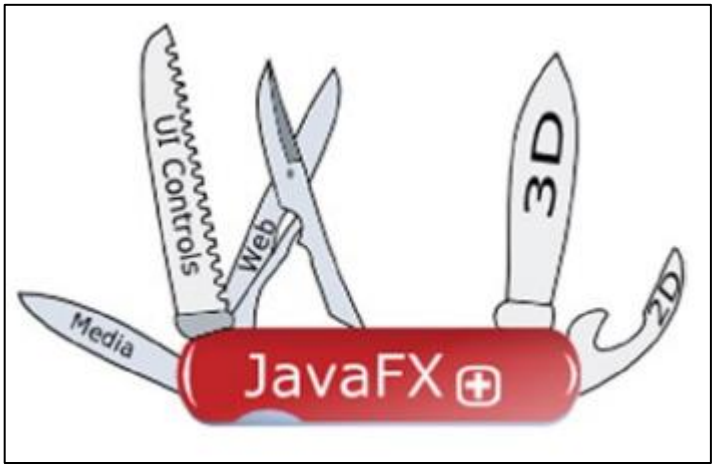
No próximo capítulo serão abordados conceitos e características específicas da linguagem JavaFX.

2.3 JAVAFX

JavaFX é uma plataforma de desenvolvimento para criação de aplicações RIA (TOPLEY, 2011), é apontada como a nova geração de kit de ferramentas Java para criação de interfaces gráficas de usuário (GUI), que permite desenvolver rapidamente aplicações ricas multiplataforma (DEA et al, 2014).

O objetivo principal do JavaFX é que a mesma aplicação possa rodar em diversos dispositivos como: smartphones, dispositivos embarcados, TVs, tablets e desktops, com poucas ou nenhuma alteração no código fonte (DEA et al, 2014). A Figura 2 faz uma analogia entre JavaFX e um canivete suíço, onde se encontra diversos recursos na mesma ferramenta.

Figura 2: JavaFX, um canivete suíço



[1] Disponível em: <http://techblog.desenvolvedores.net/2014/04/25/o-que-e-ria-rich-internet-application/> Acesso em nov. 2014.

Fonte: DEA et al, 2014

O novo JavaFX 8 é capaz de interpretar código Java puro, em GPUs (Graphics Processing Unit) modernas suporta aceleração gráfica por hardware, proporcionando uma maior riqueza de detalhes, permite aos desenvolvedores combinar animações, gráficos e controles de usuário de forma simples e eficaz (DEA et al, 2014).

2.3.1 Histórico

JavaFX teve início em 2007, criada pela Sun Microsystems, ela foi baseada em na linguagem F3 (Form Follows Function) criada em 2005 pela empresa SeeBeyond, mais especificamente por Chris Oliver (WEAVER et al, 2012). Em 2007 com a aquisição da SeeBeyond pela Sun Microsystems a linguagem F3 passou a se chamar JavaFX Script (WEAVER et al, 2012), que era uma linguagem de script declarativa, com certas semelhanças ao Java Script, e apesar de lembrar em alguns aspectos a linguagem Java, não suportava códigos Java (OLIVEIRA, 2013).

Segundo AGOSTINI e RODRIGUES, em 2009 a Sun descrevia JavaFX como:

> JavaFX fornece um modelo unificado de desenvolvimento e implantação para a construção de aplicações cliente ricas que integram mídia imersiva rica, como áudio e vídeo, gráficos, texto rico e serviços Web. JavaFX permite aos desenvolvedores de criação programarem em um contexto visual, assim, ajudando-os a trazer suas ideias para a vida mais rápido e melhor. (2010, p.137)

Apesar da grande expectativa em torno de adeptos de JavaFX, a linguagem não obteve um número expressivo de seguidores como era esperado pela Sun, tanto que em 2010 a mesma anunciou a descontinuidade da linguagem, gerando surpresa entre os que apostavam na plataforma (OLIVEIRA, 2013).

Com a aquisição da Sun Microsystems pela Oracle, em 2011 foi anunciado a retomada do projeto, anunciando o JavaFX 2.0, e com uma grande novidade, agora JavaFX era capaz de interpretar nativamente código Java, diferente da versão 1.0 que era uma linguagem declarativa de script, JavaFX 2.0 é uma linguagem puramente Java (OLIVEIRA, 2013). Desde então JavaFX vem crescendo

consideravelmente no mercado, ganhando a adesão de novos desenvolvedores (OLIVEIRA, 2013).

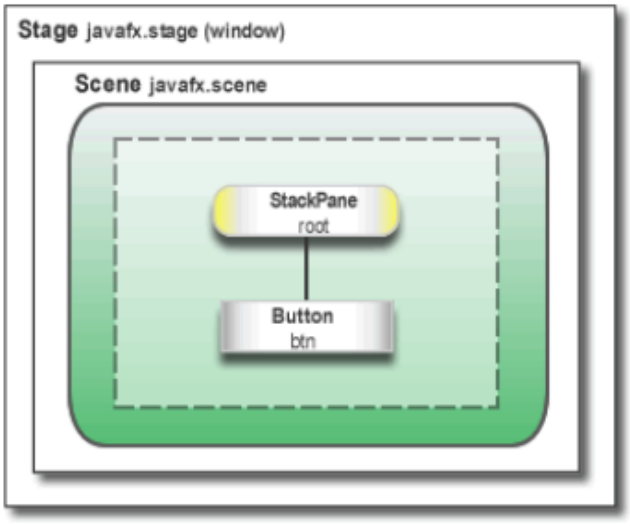
Com o lançamento da versão 2.0 de JavaFX, a Oracle declarou a linguagem como totalmente open-source (OLIVEIRA, 2013), recentemente a Oracle anunciou a versão do novo JavaFX 8 (DEA et al, 2014) que deixou de ser uma API (Aplication Programming Interface) separada, para ser nativamente parte integrante do Java 8.

2.3.2 Características

Para criar interfaces de usuários através do JavaFX é preciso entender algumas de suas características. A classe principal de um aplicativo JavaFX deve estender a classe javafx.application.Application, que deve sobrescrever o método start, que é o ponto de partida para toda a aplicação (CHAPPELL, 2013). As telas da aplicação são definidas pelos objetos *stage* e *scene*, *stage* é um elemento de nível superior que conterá os elementos *scene* que por sua vez conterá os objetos visuais da aplicação, como *buttons*, *textfields*, *datepickers* (CHAPPELL, 2013).

A Figura 3 representa o nível de hierarquia perante os elementos de uma aplicação simples, que contém uma janela com um botão com o texto “Say Hello World”. A Figura 4 mostra a aplicação em execução.

Figura 3: Hierarquia de elementos em uma aplicação JavaFX



Fonte: CHAPPELL, 2008

Figura 4: Componentes em uma aplicação compilada

Fonte: CHAPPELL, 2008

2.3.3 FXML

Há diversas formas de criar interfaces de usuário com JavaFX, é possível utilizar outras linguagens como Groovy, Scala ou Visage, e ainda instanciar os elementos *stage* e *scene* diretamente na codificação da aplicação (MARQUEZINI, 2013).

A Figura 5 mostra um exemplo de código para criação de uma interface de *login* com seus elementos criados diretamente na aplicação.

Figura 5: Componentes instanciados na aplicação

```
public class LoginApp extends Application {
    @Override
    public void start(Stage stage) throws Exception {
        AnchorPane pane = new AnchorPane();
        pane.setPrefSize(400, 300);
        TextField txLogin = new TextField();
        txLogin.setPromptText("Digite aqui seu login");
        PasswordField txSenha = new PasswordField();
        txSenha.setPromptText("Digite aqui sua senha");
        Button btEntrar = new Button("Entrar");
        Button btSair = new Button("Sair");
        pane.getChildren().addAll(txLogin, txSenha, btEntrar, btSair);
        Scene scene = new Scene(pane);
        stage.setScene(scene);
        stage.show();
    }
```

Fonte: OLIVEIRA, 2013

Além das formas citadas anteriormente, ainda é possível criar interfaces de usuários através de documentos FXML, que nada mais são do que documentos XML (Extensible Markup Language), porém documentos FXML não possuem um esquema, mas uma estrutura básica definida (FEDORTSOVA, 2014).

Criando documentos FXML para definir interfaces de usuário, a aplicação se mantém fracamente acoplada à interface, o que e uma boa prática, deixando o código organizado e facilitando manutenções futuras (FEDORTSOVA, 2014).

Arquivos FXML não são compilados com o restante da aplicação, são carregados dinamicamente à medida que a aplicação os for solicitando (WEAVER et al, 2012). Além de facilitar a criação de interfaces complexas é possível destacar alguns benefícios da utilização de documentos FXML, como apontado por FEDORTSOVA apud (MARQUEZINI, 2013, p.114)

- Facilidade no desenvolvimento, manutenção e teste das interfaces gráficas;
- O FXML não é compilado, portanto, não é necessário recompilar após modificá-lo;
- O uso de mecanismos de localização para modificar o idioma corrente é mais simples;
- O FXML pode ser usado com outras linguagens como Groovy, Scala e Clojure;
- O FXML pode ser usado como serviço, ou tarefa, ou objeto de domínio e permite o uso de linguagens de script, como o Java Script.

A Figura 6 mostra um exemplo de declaração de um arquivo FXML simples, com instância de um componente TextField e um componente Button.

Figura 6: Exemplo de documento FXML

```
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<fx:root type="javafx.scene.layout.VBox" xmlns:fx="http://javafx.com/fxml">
    <TextField fx:id="textField"/>
    <Button text="Click Me" onAction="#doSomething"/>
</fx:root>
```

Fonte: FEDORTSOVA, 2014

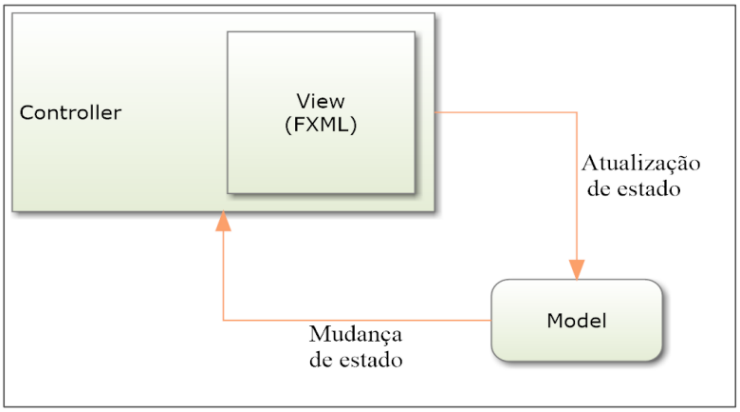
Destaca-se também na figura 6 a declaração da tag <*fx:root*>, este elemento foi adicionado a partir da versão 2.2 do JavaFX, é neste elemento que é declarado o layout da interface, seu valor é obtido pelo controller por meio do método GetRoot() do FXML Loader Class (FEDORTSOVA, 2014).

A criação de interfaces por meio de FXML é a maneira mais eficiente de explorar o modelo de desenvolvimento MVC (Model-View-Controller) traduzindo para o português: Modelo-Visão-Controle (MARQUEZINI, 2013).

Dentro da perspectiva MVC, o arquivo FXML representa a camada de Visão (View), já o Controle (Controller), é uma classe Java que geralmente implementa a classe Initializable e é declarada como controlador para o arquivo FXML (FEDORTSOVA, 2014).

A Figura 7 demonstra a interação dos elementos de uma aplicação JavaFX na perspectiva do modelo de desenvolvimento MVC.

Figura 7: JavaFX na perspectiva MVC

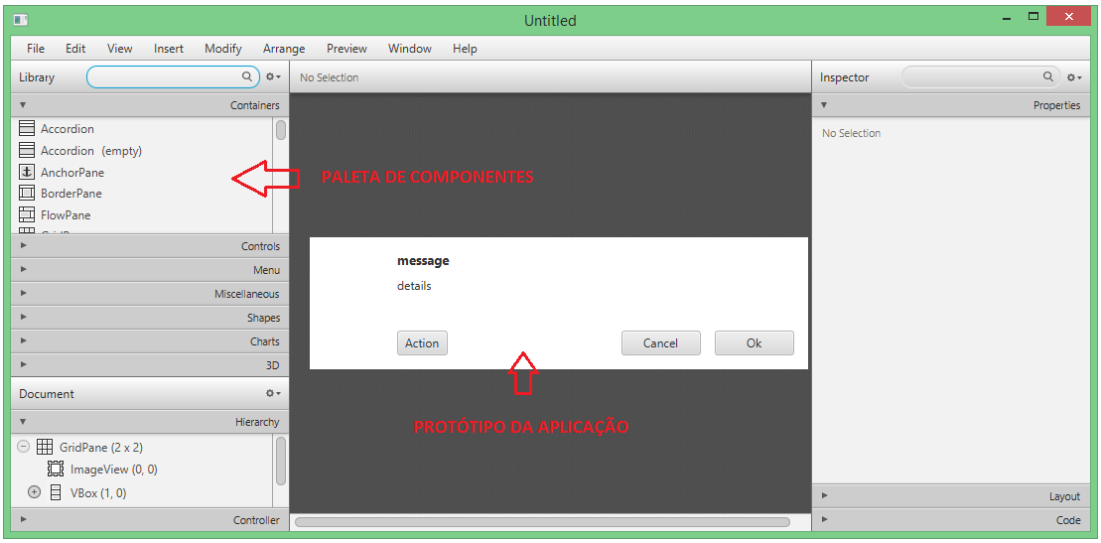


Fonte: MARQUEZINI, 2013

Outra vantagem na utilização de FXML é a possibilidade de criar interfaces por meio da ferramenta JavaFX Scene Builder. Scene Builder é uma ferramenta que permite criar interfaces de forma rápida, dinâmica e eficaz. Por meio da ferramenta é possível criar uma interface apenas arrastando os componentes disponíveis para o protótipo da aplicação exibido no centro da tela (FEDORTSOVA, 2014). O Scene Builder se encarrega de criar automaticamente o arquivo FXML com os componentes adicionados ao protótipo da aplicação (FEDORTSOVA, 2014).

A Figura 8 mostra a tela principal da ferramenta JavaFX Scene Builder, a esquerda é apresentado a paleta de componentes e no centro um protótipo de uma interface.

Figura 8: Figura 8: JavaFX Scene Builder



Fonte: Do autor, 2015

Com as características abordadas sobre JavaFX é possível concluir que a maneira mais indicada de iniciar a construção da uma interface em JavaFX é por meio de uso de FXML com a ferramenta JavaFX Scene Builder, que oferece uma maneira ágil para construção da interface e declaração dos componentes de maneira visual (arrastar componentes), além disso, JavaFX Scene Builder a partir da versão 2.0 é facilmente integrado às IDEs (Integrated Development Environment) mais utilizadas por desenvolvedores, como NetBeans e Eclipse.

2.4 THREADS

Assim como o corpo humano pode executar diversas tarefas paralelamente, ou concorrentemente, como visão, olfato, tato e paladar, os computadores também podem executar tarefas simultâneas. É comum a um computador pessoal enviar um arquivo para a impressora, executar um programa ou receber uma mensagem de correio eletrônico (DEITEL, 2010).

O Java disponibiliza Threads por meio de APIs. Ao especificar que um sistema contém Threads, significa que ele possui linhas de execução separadas,

onde cada uma possui suas pilhas de execução de métodos, e permite que sejam executadas de forma simultânea, e ainda assim compartilhar recursos do nível de aplicativo, como a memória, este recurso é denominado *multithreading* (DEITEL, 2010).

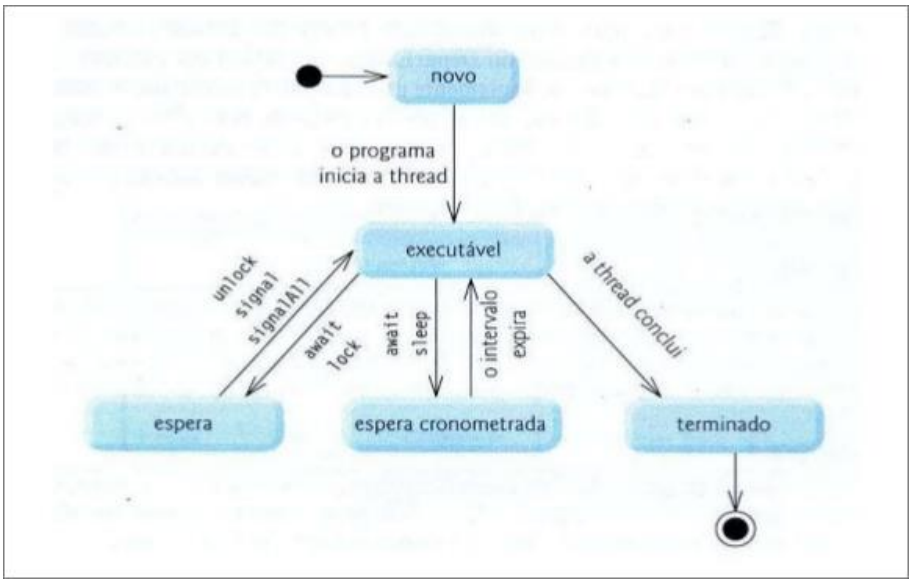
Deitel aponta diferenças entre aplicativos de uma única thread, e aplicativos com múltiplas threads:

> Um problema com aplicativos de uma única thread que pode levar a uma fraca responsividade é que as atividades longas e demoradas devem ser concluídas antes de as outras poderem iniciar. Em um aplicativo com múltiplas threads, as threads podem ser distribuídas por múltiplos processadores (se disponíveis) de modo que múltiplas tarefas sejam mesmo executadas concorrentemente e o aplicativo possa operar de modo mais eficiente. O multithreading também pode aumentar o desempenho em sistemas de um único processador que simulam a concorrência – quando uma thread não puder avançar (porque, por exemplo, está esperando o resultado de uma operação E/S), outra pode utilizar o processador. (DEITEL, 2010, p.804)

O Java inclui primitivos de multithreading, através de suas bibliotecas e recursos da própria linguagem, o que facilita a manipulação de threads de maneira portável entre diferentes plataformas, ao contrário de linguagens que não tem a capacidade de multithreading integradas, e devem realizar chamadas não portáveis aos primitivos de multithreading do sistema operacional (DEITEL, 2010).

A Figura 9 mostra o ciclo de vida de uma thread.

Figura 9: Ciclo de vida de uma thread.



Fonte: DEITEL, 2010

3 METODOLOGIA

O desenvolvimento do sistema foi realizado por meio de pesquisas bibliográficas sobre modelagem de sistemas orientados a objetos e desenvolvimento de aplicações baseadas em JavaFX.

O sistema foi modelado utilizando UML (Unified Modeling Language), respeitando os padrões da linguagem de modelagem e seguindo os requisitos levantados em um estudo de caso apresentado no próximo capítulo.

Foi utilizado PostgreSQL como SGBD (Sistema de Gerenciamento de Banco de Dados) que atualmente é um dos sistemas de código aberto mais avançado e robusto.

Para realizar a persistência de dados foi utilizado JPA, que é uma API padrão da linguagem Java para persistência de dados. O mapeamento objeto relacional foi realizado utilizando JPA, juntamente com o provedor de persistência Hibernate.

A aplicação foi desenvolvida com a linguagem JavaFX, que vem crescendo consideravelmente após o lançamento da versão 2.0 e, atualmente na versão 8, passou a ser parte integrante do Java.

3.1 ESTUDO DE CASO

Esta seção apresenta o estudo de caso realizado. Aborda o histórico da empresa, da marca que ela representa e uma descrição do processo atual. É apresentada uma breve análise dos principais problemas encontrados no processo.

3.1.1 Histórico da Empresa

A empresa Gambatto Auto Ltda. atua como concessionária de veículos, representante da marca Ford para Passo Fundo e região, líder de mercado na região norte do Estado do Rio Grande do sul, iniciou suas atividades em 2003, na cidade de Passo Fundo. A mesma possui showroom de veículos novos e seminovos atuando também na área de pós-vendas. Atualmente possui uma grade com aproximadamente 80 funcionários, distribuídos entre a Matriz de Passo Fundo e Filial em Erechim.

A empresa conta com um sistema de DMS (Dealer Management System) para controle financeiro, gerencial e administrativo. Conta com servidores próprios situados na matriz e uma rede privada (VPN) para interligação com a Filial e acesso externo aos servidores.

3.1.2 Contexto Empresarial

a) O Processo Atual

Ford Motor Company é uma empresa Norte Americana fundada por Henry Ford em 1903 na cidade de Detroit, Estados Unidos, foi também a primeira indústria automobilística a se instalar no Brasil, em 1929, na cidade de São Paulo, atualmente possui quatro fábricas instaladas no Brasil, sendo três em São Paulo, nas cidades de São Bernardo do Campo, Taubaté e Tatuí, este conta também com um dos mais modernos campo de provas do mundo, utilizado como base nos testes dos veículos fabricados pela montadora, e uma fábrica situada no estado da Bahia, na cidade de Camaçari, com capacidade de produzir até 250 mil veículos por ano (FORD, 2014).

Basicamente a produção da montadora é dividida em quatro segmentos: Carros, SUVs & Crossovers, Pick-Ups e Caminhões. Visando melhor atender o público alvo, a montadora dividiu estes segmentos em duas linhas distintas de distribuidores, o segmento de caminhões passou a ser um distribuidor distinto dos demais, atendendo especificamente a linha pesada da montadora, basicamente comercial. Os demais segmentos são atendidos por outra linha de distribuidores, popularmente conhecidos como veículos leves e/ou de passeio.

A empresa Gambatto Auto Ltda. é uma concessionária, ou também conhecida como distribuidora, representante da linha de veículos leves, em seu showroom de veículos novos é encontrado toda a linha do segmento de Carros, composto pelos modelos: Novo Ka, Novo Ka+, New Fiesta Hatch, New Fiesta Sedan, Novo Focus Hatch, Novo Focus Sedan e Fusion, também o segmento de SUVs e Crossovers composto pelos modelos: Ecosport e Edge, e finalmente pelo segmento de Pick-Ups, composto pelos modelos: Ranger Cabine Simples e Ranger Cabine Dupla.

O setor de venda de veículos da empresa conta com uma equipe composta atualmente por onze vendedores e uma recepcionista na sua Matriz em Passo

Fundo - RS, e uma equipe composta por cinco vendedores e uma recepcionista na sua Filial de Erechim – RS.

A equipe de vendedores é coordenada por dois gerentes de venda, sendo um responsável pela unidade de Passo Fundo e outro responsável pela unidade de Erechim, estes gerentes são coordenados pelo gerente geral da Empresa, responsável por traçar as metas e objetivos de venda, que são repassadas aos gerentes de venda, e estes por sua vez, repassam aos seus vendedores.

As vendas realizadas pela empresa basicamente são iniciadas das seguintes formas:

- O cliente realiza uma visita ao distribuidor.
- O cliente realiza um contato telefônico ao distribuidor.
- O vendedor contata o cliente por telefone por meio de sua lista de contatos.

Se o cliente realizar uma visita ao distribuidor a recepcionista registra alguns dados do cliente em uma planilha (Anexo A) que posteriormente é repassada ao Gerente de Vendas. A planilha apenas é arquivada, não é realizado nenhum tipo de controle com os dados contidos nela. Após o preenchimento desta planilha o cliente e encaminhado até um vendedor que esteja disponível.

Quando realizado o primeiro contato, seja qual forma citada anteriormente, se dá início a uma série de contatos entre cliente e vendedor a fim de tratar os detalhes do veículo em negociação como: modelo, cor, tipo de pintura, preço, forma de pagamento, etc.

Entre os diversos contatos realizados entre vendedor e cliente é muito comum haver contatos entre vendedor e gerente de vendas, a fim de explanar os detalhes dos negócios em andamento, e em alguns casos, o vendedor necessita da aprovação do seu gerente de vendas para efetuar a venda com condições especiais que fogem do poder de decisão direta do vendedor, seja por valor do veículo vendido, forma de pagamento, valor do recebimento do veículo seminovo, etc. Pode haver eventualmente contatos entre gerente de vendas e gerente geral para aprovação de condições especiais que fogem do poder de decisão do gerente de vendas.

Os contatos são realizados de forma verbal ou via e-mail, em alguns casos os vendedores realizam anotações sobre os contatos por eles realizados, seja em uma agenda pessoal, em um documento de texto em seu computador, anotações em seus smartphones ou até mesmo pequenas anotações em papéis de rascunho. Seja qual for o meio de controle, ele é apenas de uso pessoal do vendedor.

Se um cliente solicitar um orçamento do veículo em negociação é entregue a ele pelo vendedor um folder do veículo em questão (se este veículo possuir um folder), e no verso deste, escrito à mão as informações que contemplam o orçamento e o nome do vendedor responsável, ou então é entregue ao cliente uma folha impressa com as informações do veículo obtidas no site da montadora e também constando no verso, escrito à mão, as informações complementares e os dados do vendedor responsável.

Caso haja concordância nos termos da negociação entre as partes envolvidas, o vendedor é autorizado pelo gerente de vendas a efetuar a venda do veículo ao cliente. O vendedor inicia então o preenchimento de um termo de negociação que pode ser visualizado no anexo B, que é um documento onde constam as informações:

- Dados do veículo vendido;
- Informações dos acessórios adicionais (caso o cliente opte por algum acessório adicional que não acompanha originalmente o veículo escolhido);
- Dados cadastrais do cliente;
- Descrição da forma de pagamento;
- Informações do veículo recebido (caso o cliente opte por entregar um veículo seminovo como parte do pagamento);
- Informações sobre a instituição financeira (caso o cliente opte por financiar o valor do veículo adquirido, seja total ou parcial);
- Informações do vendedor que realizou a venda.

Com o termo de negociação devidamente preenchido e assinado pelo cliente, vendedor e gerente de vendas, este é repassado ao setor de faturamento para emissão da Nota Fiscal de venda.

b) Análise do Processo Atual

Segundo a descrição do processo é possível observar que a empresa não possui qualquer tipo de controle dos contatos realizados por seus vendedores. A única informação disponível para a empresa é os dados que constam no termo de negociação, ou seja, na empresa está apenas a informação das negociações das vendas concretizadas por seus vendedores. Esta informação é completamente resumida, pois é composta apenas por dados referentes aos componentes da negociação, não há qualquer registro dos diversos contatos realizados até se concretizar a venda.

Manter todos os detalhes de uma negociação é crucial para a empresa tomar conhecimento de como tal negociação foi conduzida pelo vendedor. Com tal informação também é possível analisar o desempenho da sua equipe de vendas e também traçar um perfil de seus clientes.

Na próxima seção será apresentada a modelagem do sistema baseado no estudo de caso, capaz de auxiliar na organização no processo de atendimento a clientes da concessionária.

3.2 MODELAGEM DO SISTEMA

Seguindo os requisitos levantados no estudo de caso, foi realizado a modelagem do sistema, este tem como objetivo manter um histórico dos atendimentos realizados pelos vendedores da empresa Gambatto Auto Ltda. Visa resolver de forma parcial os problemas encontrados no processo atual da empresa.

3.2.1 Requisitos Funcionais

Requisitos funcionais definem as funções que o sistema deverá oferecer aos seus usuários. Com base no estudo de caso foram levantados os seguintes requisitos funcionais que o sistema deverá atender.

- **RF1.** O sistema deve manter cadastro de usuários, bem como as suas permissões de acesso a funções especiais da aplicação;

- **RF2.** O sistema deve manter o registro dos atendimentos realizados pelos vendedores da empresa, bem como o status de como o mesmo foi encerrado, como venda efetuada ou como venda perdida;
- **RF3.** O sistema de manter o registro dos contatos realizados entre vendedor e cliente, para cada atendimento realizado. Um atendimento pode conter diversos contatos, até que o mesmo seja encerrado;
- **RF4.** O sistema deve manter um cadastro de acessórios disponíveis para um determinado veículo. A lista de acessórios é classificada por modelo de veículo. A lista de modelos será obtida do sistema de Banco de Dados já existente, assim como a lista de veículos em estoque;
- **RF5.** O sistema deve permitir a emissão de orçamentos impressos ou enviados via e-mail, contendo os detalhes do veículo em negociação. O cliente pode optar por receber o orçamento via e-mail e impresso, se desejar;
- **RF6.** O sistema deve permitir a abertura de um atendimento para novos clientes, apenas informado o seu nome e telefone para contato, os demais dados serão exigidos caso a venda do veículo seja concretizada, a fim de atualizar os dados cadastrais do cliente e enviar a solicitação de emissão de Nota Fiscal para o setor de faturamento.

3.2.2 Requisitos Não Funcionais

Requisitos não funcionais estão ligados ao uso da aplicação como usabilidade, tecnologias envolvidas e desempenho. Com base no estudo de caso foram levantados os seguintes requisitos não funcionais que o sistema deverá atender.

- **RNF1.** A aplicação deve ser desenvolvida com a linguagem JavaFX, a ser executado corretamente independente do sistema operacional do usuário;
- **RNF2.** O sistema deverá permitir a sua execução de forma local (modo desktop) ou de forma remota (via web), para diminuir o impacto e

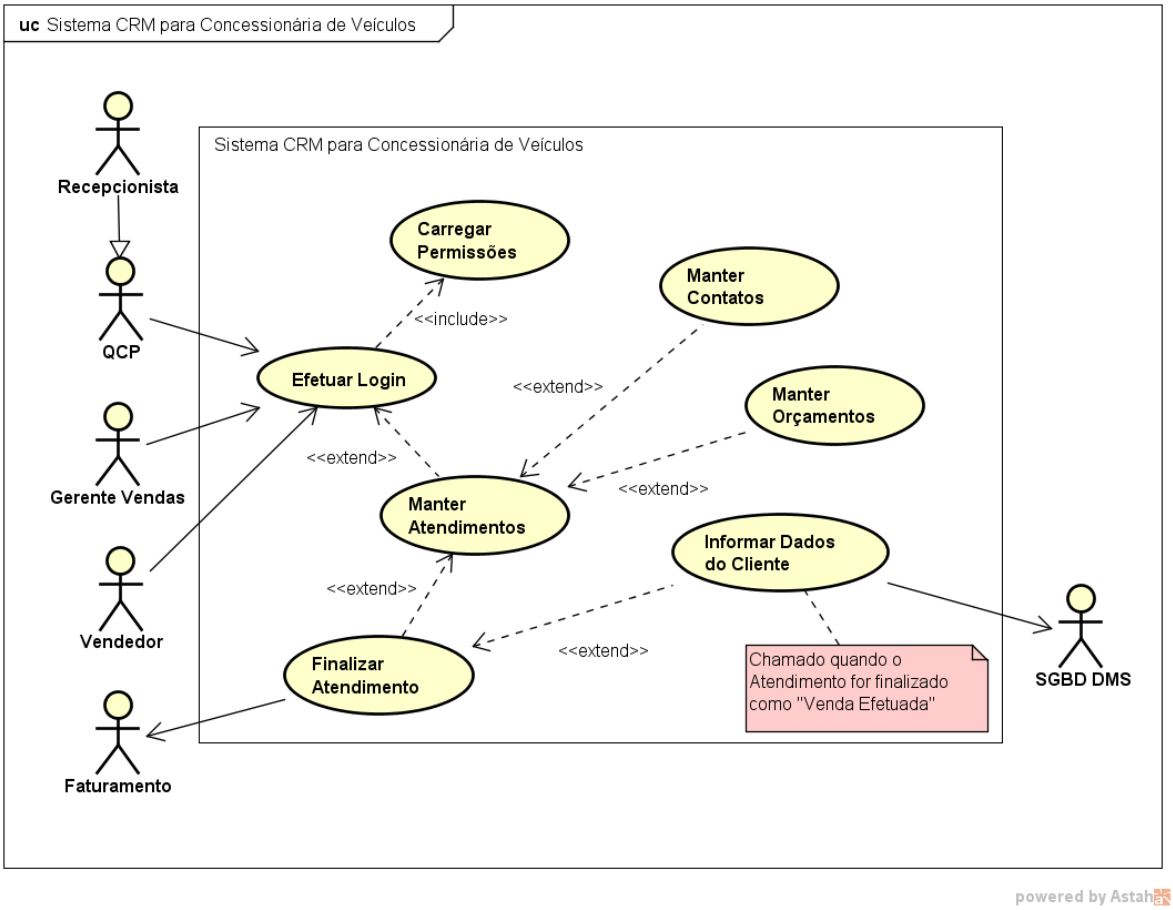
atender a necessidade de acesso interno e externo ao ambiente da empresa;

- **RNF3.** Será utilizado o PostgreSQL como SGBD;
- **RNF4.** Deverá ser utilizado JPA com framework Hibernate para realizar a persistência dos dados e o mapeamento objeto-relacional respectivamente;
- **RNF5.** A interface do sistema deve ser agradável e objetiva, a fim de facilitar ao usuário a interação com as funcionalidades do sistema;
- **RNF6.** O sistema fará uso de *login* e senha para registrar o acesso dos usuários;
- **RNF7.** Os Usuários deverão possuir permissões para acesso às funcionalidades do sistema.
- **RNF8**. O sistema deve ser confiável, e atender aos requisitos levantados.

3.3 DIAGRAMA DE CASO DE USO

O diagrama de caso de uso tem como objetivo demonstrar como os usuários (atores) irão interagir com os processos (casos de uso) do sistema proposto. A Figura 10 representa os casos de uso levantados, baseados no estudo de caso realizado.

Figura 10: Diagrama de Casos de Uso



Fonte: Do autor, 2015

O ator QCP (Qualidade Compromisso e Participação) representa os usuários do setor de qualidade da empresa, ele pode realizar *login* no sistema e realizar consulta de Atendimentos, além de poder informar novos Atendimentos e direcioná-los a um Vendedor. Pode também informar novos contatos para os Atendimentos com status: “Aberto”.

A Recepcionista é um ator filho do QCP, com isso, herda todos os seus casos de uso.

O Gerente de Vendas e o Vendedor podem realizar *login* no sistema, realizarem consulta e informar novos Atendimentos. Podem também informar novos contatos para os Atendimentos com status: “Aberto”, emitir orçamentos para estes e finalizar tais Atendimentos e enviá-los ao Faturamento.

O Gerente de Vendas pode também realizar a troca do Vendedor vinculado a um Atendimento e realizar a edição dos dados do cliente, como nome, telefone e e-mail.

O Vendedor, ao finalizar um Atendimento como: “Venda Efetuada” deve realizar a atualização dos dados cadastrais do Cliente, representado pelo caso de uso Informar Dados do Cliente, estes dados são enviados ao SGBD DMS e após o processo é entregue ao setor de Faturamento para emissão das Notas Fiscais.

A Seguir é apresentada a documentação de cada caso de uso levantado.

3.4 DOCUMENTAÇÃO DOS CASOS DE USO

A documentação dos casos de uso visa mostrar uma visão mais detalhada do diagrama de casos de uso. São imprescindíveis para o processo de desenvolvimento da aplicação.

A seguir é apresentado o detalhamento dos casos de uso apresentados no diagrama da Figura 10. Nos Cenários Principais e Secundários, as referências de Usuário remetem aos atores listados na seção Atores de cada caso de uso.

3.4.1 Caso de Uso Efetuar *Login*

Tabela 1: Documentação do Caso de Uso Efetuar *Login*

UC001	**Efetuar Login**
Breve Descritivo: Este Caso de Uso descreve o processo para efetuar *login* no sistema.	
Pré-Condições: Aplicação iniciada.	
Atores: QCP, Recepcionista, Gerente Vendas, Vendedor.	
Cenários Principais:	
1 – A aplicação informa ao usuário um campo com a data a ser utilizada para realização das operações no sistema. O campo é baseado na data do Sistema Operacional do Usuário.	
2 – Usuário informa *login* e senha desejada. O *login* e senha devem conter letras e números e ter no mínimo seis caracteres.	
3 – Aplicação verifica a existência do *login* e senha informados no banco de dados, se os dados informados consistirem com os cadastrados no banco dados a execução do sistema é autorizada, caso contrário emite um alerta ao Usuário informando o *login* ou senha incorretos.	

Cenários Alternativos:
2.1 – Não aceita texto com menos de três caracteres. Alerta o Usuário.
2.2 – Caso o *login* já esteja em uso, emitir um alerta ao Usuário.
2.3 – Caso seja identificado mais de três tentativas de *login* incorreto, encerrar a aplicação.
Requisitos Especiais: 1. Deve haver uma função para controlar os usuários que tenham permissão de efetuar *login* no sistema com data inferior a data atual do Bando de Dados.
Dados: Login: Texto 20 Senha: Texto 20
Analista de Negócio: O Autor
Versão: 1.0

3.4.2 Caso de Uso Carregar Permissões

Tabela 2: Documentação do Caso de Uso Carregar Permissões

UC002	**Carregar Permissões**
Breve Descritivo: Este Caso de Uso descreve o processo de carregamento das permissões do Usuário.	
Pré-Condições: *Login* efetuado com sucesso.	
Atores: QCP, Recepcionista, Gerente Vendas, Vendedor.	
Cenários Principais:	
1 – Carrega a lista de menus liberados ao Usuário.	
2 – Carrega a lista de funções liberadas ao Usuário.	
3 – Apresenta a tela principal da aplicação baseada nas permissões do Usuário.	
Observação: Prever Caso de Uso **Efetuar *Login***.	
Analista de Negócio: O Autor	
Versão: 1.0	

3.4.3 Caso de Uso Manter Atendimento

Tabela 3: Documentação do Caso de Uso Manter Atendimento

UC003	**Manter Atendimento**
Breve Descritivo: Este Caso de Uso descreve o processo de manutenção dos atendimentos a clientes.	
Pré-Condições: *Login* efetuado com sucesso. Permissões do Usuário carregadas.	
Atores: QCP, Recepcionista, Gerente Vendas, Vendedor.	
Cenários Principais:	
1 – O Usuário informa os dados do Cliente em atendimento: nome, telefone e e-mail.	
2 – O Usuário seleciona um vendedor a ser vinculado ao atendimento.	
3 – A aplicação informa um campo com a data, baseada na data informada ao realizar o *login* no sistema, para registrar como data de abertura do atendimento.	
4 – A aplicação insere os valores informados no Banco de Dados	
Cenários Alternativos:	
1.1 – O Sistema realiza um busca no banco de dados por atendimentos baseados no telefone e e-mail informados.	
2.1 – Caso o sistema encontre atendimentos anteriores cadastrados, realiza uma comparação do vendedor referenciado no último atendimento cadastrado com o vendedor informado no atendimento atual. Se forem identificados vendedores diferentes, é emitido um alerta ao Usuário com os dados do último atendimento. Emite pergunta ao Usuário se deseja alterar o vendedor selecionado para o atendimento atual.	
Requisitos Especiais: 2. Deve haver uma função para controle dos Usuários com permissão para alterar o vendedor a ser vinculado no atendimento. 3. Deve haver uma função para controle dos Usuários com permissão para cadastrar um atendimento com vendedor diferente do último atendimento cadastrado. 4. O atendimento aberto conterá um campo status, onde poderá conter as situações: “A” para atendimento aberto, “C” para atendimento cancelado, “F” para atendimentos em processo de faturamento (emissão das notas fiscais), “P” para atendimento que não foi efetuado a venda (venda perdida) e “V” para atendimento em que a venda foi concretizada.	
Dados: Nome Cliente: Texto 60 Telefone Cliente: Texto 10	

E-mail Cliente: Texto 60 Status: Char 1
Observação: Prever Caso de Uso **Efetuar *Login*, Carregar Permissões**.
Analista de Negócio: O Autor
Versão: 1.0

3.4.4 Caso de Uso Manter Contato

Tabela 4: Documentação do Caso de Uso Manter Contato

UC004	**Manter Contato**
Breve Descritivo: Este Caso de Uso descreve o processo de manutenção dos contatos realizados com clientes da concessionária referente a um atendimento aberto.	
Pré-Condições: *Login* efetuado com sucesso. Permissões do Usuário carregadas. Deve haver um atendimento aberto que será referenciado pelo contato.	
Atores: QCP, Recepcionista, Gerente Vendas, Vendedor.	
Cenários Principais:	
1 – O Usuário seleciona um atendimento que esteja criado.	
2 – O Usuário informa uma descrição do que foi tratado com o cliente.	
3 – A aplicação informa um campo com a data, baseada na data informada ao realizar o *login* no sistema, para registrar como data de realização do contato.	
Cenários Alternativos:	
1.1 – Permite que sejam selecionados apenas atendimentos que estejam com o status “A”.	
2.1 – Não permite descrições com menos de 10 e mais de 500 caracteres.	
Requisitos Especiais: 1. Deve haver uma função para controle dos Usuários com permissão para informar contatos referentes a outros usuários.	
Dados: Descrição do contato: Texto 500 Data do contato: Data (DD/MM/AA)	
Observação: Prever Caso de Uso **Efetuar *Login*, Carregar Permissões, Manter**	

Atendimento.
Analista de Negócio: O Autor
Versão: 1.0

3.4.5 Caso de Uso Emitir Orçamento

Tabela 5: Documentação do Caso de Uso Manter Orçamentos

UC005	**Emitir Orçamento**
Breve Descritivo: Este Caso de Uso descreve o processo de emissão de orçamentos para um contato realizado.	
Pré-Condições: *Login* efetuado com sucesso. Permissões do Usuário carregadas. Atendimento aberto. Contato informado.	
Atores: Gerente Vendas, Vendedor, SGBD DMS.	
Cenários Principais:	
1 – A Aplicação mostra uma lista de veículos presentes no estoque da concessionária. Esta lista será oriunda do bando de Dados do sistema de DMS atual.	
2 – O Usuário seleciona o veículo de interesse do cliente.	
3 – O Usuário informa uma lista de acessórios e seus respectivos valores, que o cliente deseja acrescentar ao veículo (opcional).	
4 – O Usuário informa uma observação ao orçamento (opcional) e um valor concedido como desconto (opcional) e a aplicação calcula o valor total do orçamento com base no valor do veículo selecionado, somando o valor dos acessórios e abatendo o valor do desconto concedido.	
4 – O usuário emite o orçamento para o cliente.	
5 – A Aplicação grava em seu Banco de Dados os dados do veículo importados do Banco de Dados do sistema de DMS.	
Cenários Alternativos:	
1.1 – A lista será preenchida com veículos cadastrados no Banco de Dados do DMS que estiverem com status: “TM” trânsito Montadora, “ES” estoque, “TF” trânsito filial e “RE” reservado.	
1.2 – Veículos com status “RE” devem aparecer em vermelho na lista de estoque.	

2.1 – Ao selecionar um veículo com status “RE”, emitir um aviso ao Usuário.
3.1 – A lista de acessórios deve estar previamente cadastrada no sistema com seus respectivos valores.
Dados: Valor de desconto: Double Valor dos acessórios: Double Valor total do orçamento: Double Observação: Texto 200 Descrição do acessório: Texto 100 Valor do acessório: Double
Observação: Prever Caso de Uso **Efetuar *Login*, Carregar Permissões, Manter Atendimento, Manter Contato**.
Analista de Negócio: O Autor
Versão: 1.0

3.4.6 Caso de Uso Finalizar Atendimento

Tabela 6: Documentação do Caso de Uso Finalizar Atendimento

UC006	**Finalizar Atendimento**
Breve Descritivo: Este Caso de Uso descreve o processo de encerramento de um atendimento.	
Pré-Condições: *Login* efetuado com sucesso. Permissões do Usuário carregadas. Atendimento aberto.	
Atores: QCP, Recepcionista, Gerente Vendas, Vendedor.	
Cenários Principais:	
1 – O Usuário seleciona um atendimento.	
2 – Usuário finaliza o atendimento selecionando status: “C” para cancelar um atendimento, “P” para informar uma venda perdida e “V” para informar um atendimento com venda concretizada.	
3 – A Aplicação realiza a alteração do atendimento no Banco de Dados.	
Cenários Alternativos:	
1.1 – Exibe uma lista com atendimentos com status “A”. Permitir filtrar atendimentos	

de um determinado vendedor e de um determinado período.
Requisitos Especiais: 1. Deve haver uma função para controlar os usuários que tenham permissão para finalizar atendimentos de outros usuários. 2. Deve registrar o usuário que finalizou o atendimento
Dados: Status: Char 1
Analista de Negócio: O Autor
Versão: 1.0

3.4.7 Caso de Uso Informar Dados do Cliente

Tabela 7: Documentação do Caso de Uso Informar Dados do Cliente

UC007	**Informar Dados do Cliente**
Breve Descritivo: Este Caso de Uso descreve o processo de atualização cadastral do cliente quando este efetuar uma nova compra.	
Pré-Condições: Login efetuado com sucesso. Permissões do Usuário carregadas. Atendimento aberto. Contatos informados. Atendimento finalizado com status “V”	
Atores: Gerente de Vendas, Vendedor.	
Cenários Principais:	
1 – O usuário informa os dados do cliente.	
2 – A Aplicação atualiza os dados cadastrais do cliente em seu Banco de Dados e no Banco de Dados do sistema DMS.	
Dados: Nome: Texto 100 Endereço: Texto 100 Número: Texto 10 Complemento: Texto 30 Bairro: Texto: 100 Cep: Texto 8 **Pessoa Física** Número do RG: Texto 15	

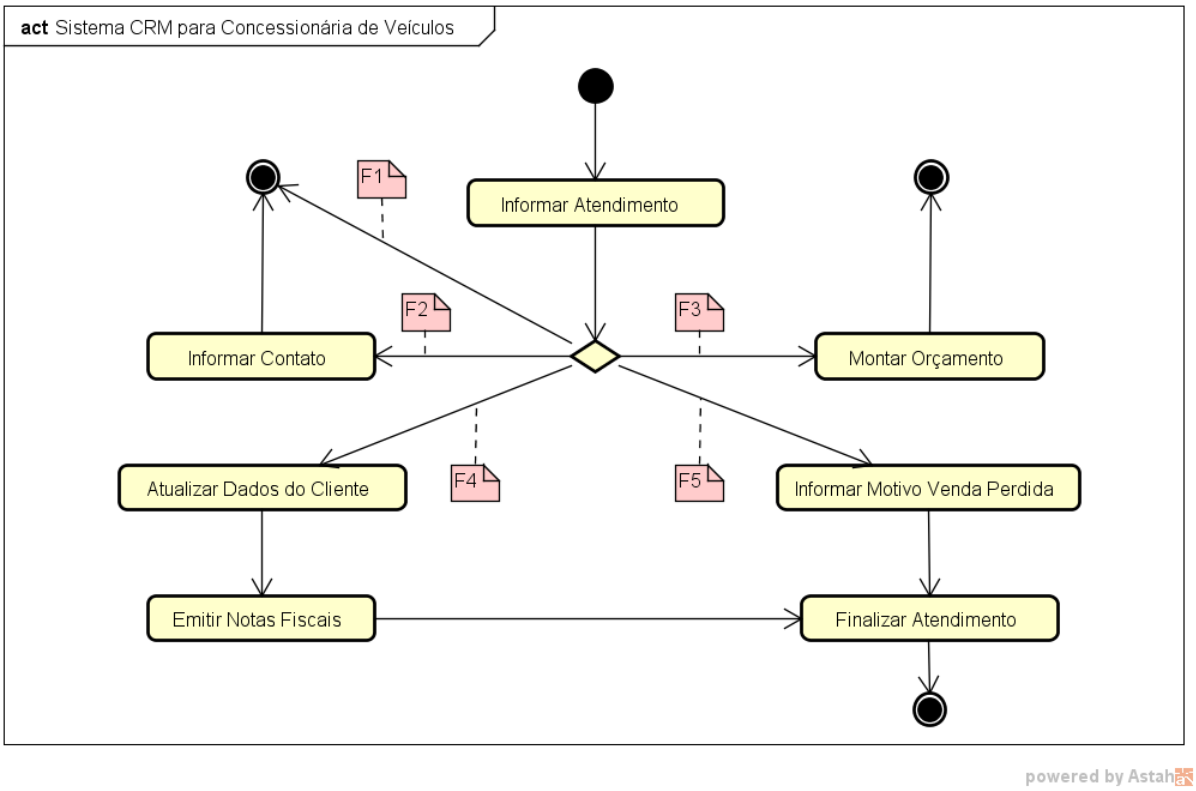
Emissor do RG: Texto 10 Número do CPF: Texto 11 Data de Nascimento: Date (DD/MM/AAA) Estado Civil: Char 1 Profissão: Texto 30 **Pessoa Jurídica** Nome Fantasia: Texto 100 Número do CNPJ: texto 14 Número da Inscrição Estadual: Texto 14
Analista de Negócio: O Autor
Versão: 1.0

3.5 DIAGRAMA DE ATIVIDADES

O diagrama de atividades, diferente do diagrama de casos de uso, visa dar uma visão segmentada do fluxo do sistema. Demonstra o fluxo das atividades, que possuem um ponto de início e fim, passando por suas atividades intermediárias.

A Figura 11 demostra o fluxo das atividades baseadas no estudo de caso realizado. O fluxo contém um ponto de partida e diversos pontos de saída possíveis, conforme fluxo seguido pelo usuário.

Figura 11: Diagrama de Atividades



Fonte: Do autor, 2015

O Diagrama de Atividades acima é focado na principal atividade do sistema, que é o controle dos atendimentos realizados. Para descrição dos fluxos foi criados os identificadores: F1, F2, F3, F4 e F5. Abaixo um breve descritivo de cada um dos Fluxos.

- **F1.** Este fluxo se encerra após informar um atendimento. É possível exemplificar neste caso, o atendimento realizado pela recepcionista de vendas, onde ela realiza o primeiro atendimento ao cliente, informa este no sistema e encaminha o cliente para o vendedor informado por ela no atendimento, para que este dê continuidade no atendimento.
- **F2.** Este fluxo identifica a informação dos contatos realizados para o atendimento. Neste fluxo o atendimento não é encerrado, identifica que a negociação ainda está em andamento.
- **F3.** Este fluxo identifica a emissão de um orçamento para o atendimento.

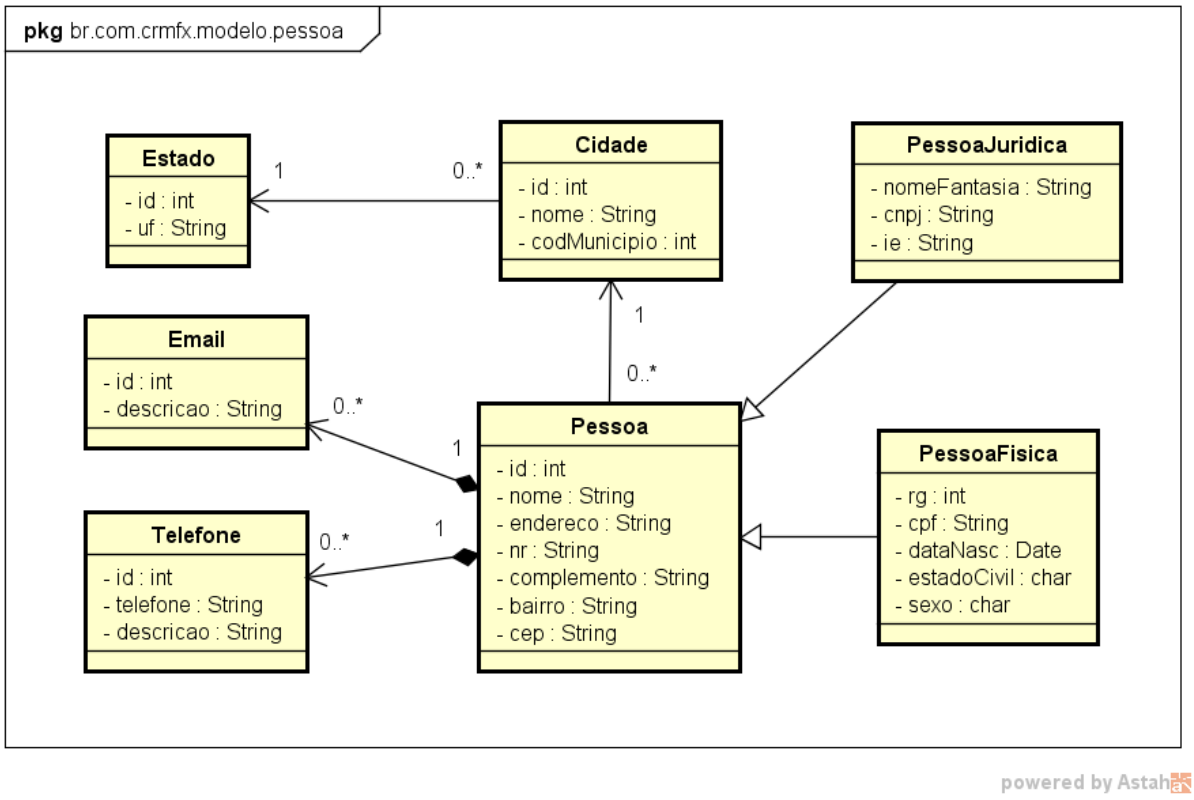
- **F4.** Este fluxo identifica o inicio do processo de venda de um veículo, onde o vendedor atualiza os dados cadastrais do cliente no atendimento e encaminha este para o setor de faturamento, para emissão das notas fiscais e finalização do atendimento. Quando o vendedor encaminha o atendimento para faturamento, este passa do status “A” (aberto) para o status “F” (pendente de faturamento). Após a emissão das notas fiscais, o faturista encerra o atendimento com o status “V” (venda efetuada).
- **F5.** Este fluxo identifica o encerramento de um atendimento com o status “P” (venda perdida). Para efetuar o encerramento como venda perdida, o usuário deverá informar o motivo da perda desta venda.

3.6 DIAGRAMA DE CLASSES

O diagrama de classes tem como objetivo demonstrar como as classes são organizadas no sistema. São as classes utilizadas para a persistência de dados. O diagrama de classes segundo Guedes apud (Bringhenti, 2015, p.37) “mostra de maneira estática como as classes estão organizadas e define suas estruturas lógicas”.

Para uma melhor descrição, o diagrama de classes foi desmembrado no módulo Pessoa, módulo Revenda e módulo Atendimento, conforme segue:

Figura 12: Diagrama de Classes, Módulo Pessoa

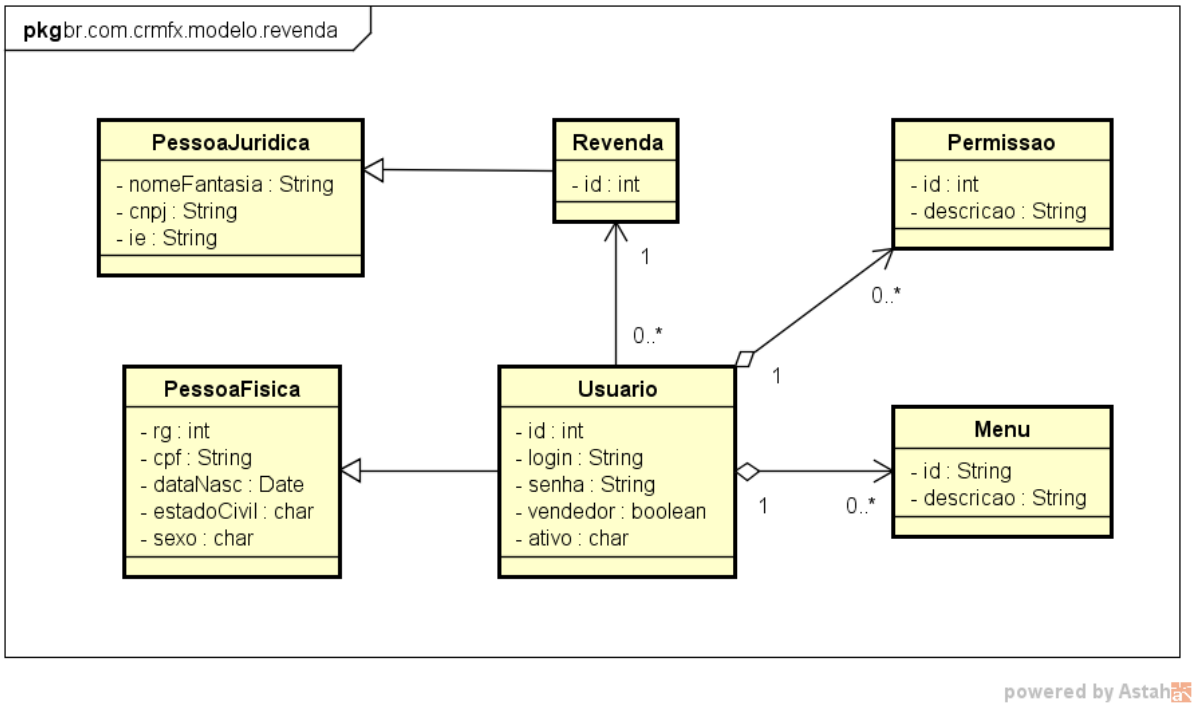


Fonte: Do autor, 2015

O objetivo da classe Pessoa é armazenar os dados relativos às pessoas, ela é abstrata, mantém os dados comuns as suas duas classes filhas – PessoaFisica e PessoaJuridica – logo, estas classes herdam todos os seus atributos. A classe Pessoa também possui duas listas, da classe Email e da classe Telefone, onde uma Pessoa pode não ter nenhum ou muitos e-mails, e pode ter nenhum ou muitos telefones.

A classe Cidade se relaciona com as classes Pessoa e Estado. No relacionamento com a classe Pessoa, uma pessoa deve referenciar uma cidade, e uma cidade pode ser referenciada por muitas pessoas. No relacionamento com a classe Estado, uma cidade deve referenciar um estado e um estado pode ser referenciado por muitas cidades.

Figura 13: Diagrama de Classes, Módulo Revenda



Fonte: Do autor, 2015

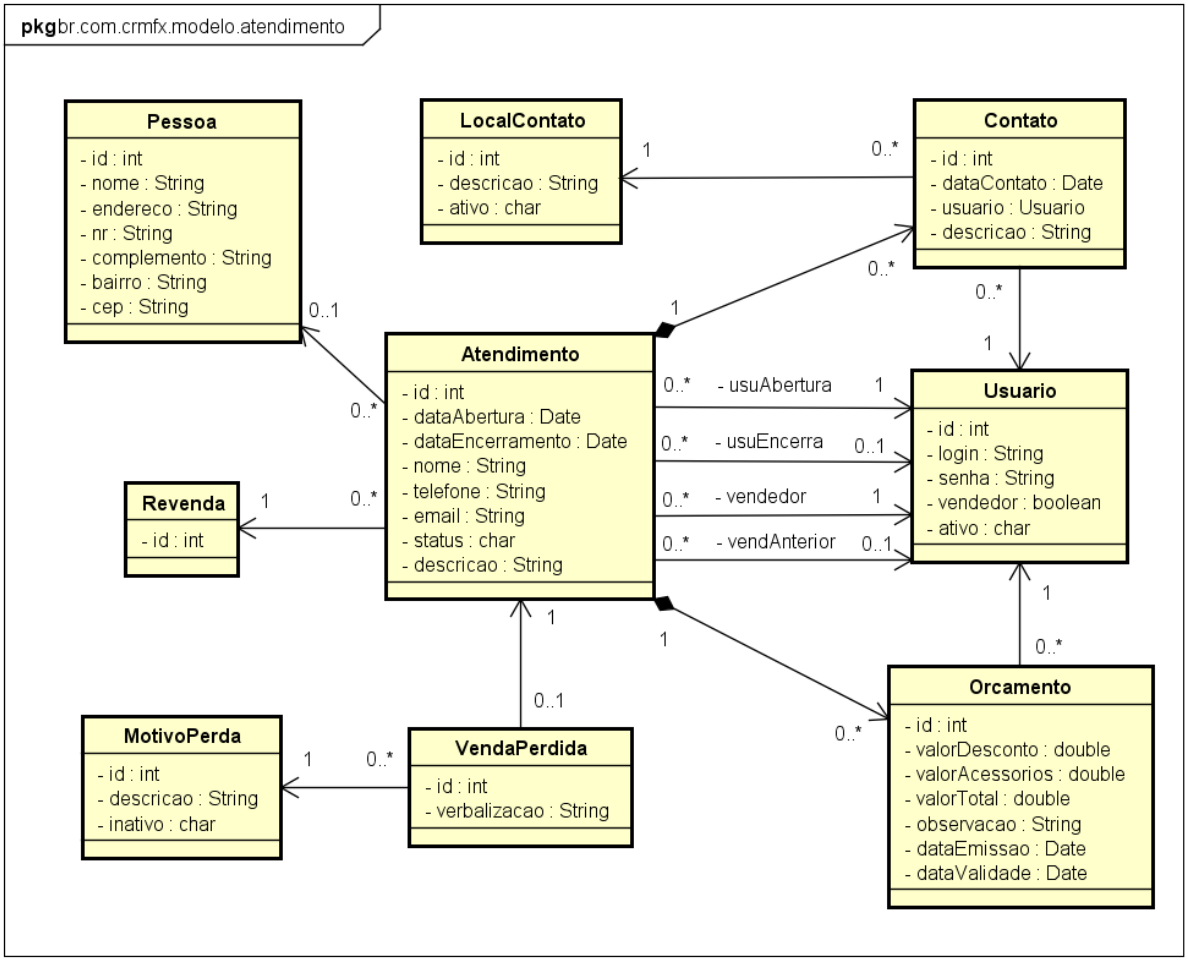
A classe Usuario é filha da classe PessoaFisica, logo, herda os seus atributos, ela é responsável por manter as informações dos usuários que possuem acesso ao sistema e possui também duas listas, uma da classe Menu e outra da classe Permissao.

Cada usuário deverá ter uma lista de menus liberados em seu perfil de acesso. A classe Menu é responsável por manter os menus liberados para o usuário, onde um usuário pode não ter nenhum ou muitos menus.

Já a classe Permissao é responsável por manter as permissões de acesso do usuário aos módulos do sistema, onde um usuário pode não ter nenhuma ou muitas permissões.

A identificação da concessionária é realizado na classe Revenda, que é filha da classe PessoaJuridica. O seu atributo “id” é informado pelo usuário no momento do cadastro da revenda. Este será o código que identificará a revenda no sistema. Ela possui um relacionamento com a classe Usuario, onde um usuário deverá referenciar uma revenda e uma revenda poderá ter nenhum ou muitos usuários. A revenda informada para o usuário será utilizada como a revenda padrão do usuário, quando este efetuar o *login* no sistema.

Figura 14: Diagrama de Classes, Módulo Atendimento



Fonte: Do autor, 2015

A classe Atendimento é responsável por manter os dados dos atendimentos realizados, ela possui quatro relacionamentos com a classe Usuario.

O relacionamento identificado como “usuAbertura” identifica o usuário responsável pela abertura do atendimento, um atendimento deve ter um usuário e um usuário pode ter nenhum ou muitos atendimentos.

Já o relacionamento identificado como “usuEncerra” identifica o usuário responsável pelo encerramento do atendimento, um atendimento poderá ter nenhum ou no máximo um usuário.

O terceiro relacionamento identificado como “vendedor” identifica o vendedor responsável pelo atendimento, um atendimento deve ter um vendedor e um vendedor poderá ter nenhum ou muitos atendimentos.

Por fim o relacionamento identificado como “vendAnterior” identifica o vendedor responsável por atendimentos anteriores a este cliente, se for o caso. Um atendimento pode possuir nenhum ou no máximo um vendedor, e um vendedor poderá ter nenhum ou muitos atendimentos.

A classe Orcamento é responsável por manter os dados dos orçamentos emitidos, ela possui relacionamentos com as classes Atendimento e Usuário, onde um orçamento deverá referenciar um Atendimento, e um atendimento poderá ter nenhum ou muitos orçamentos. Um orçamento deverá referenciar um usuário, e um usuário poderá ter nenhum ou muitos orçamentos.

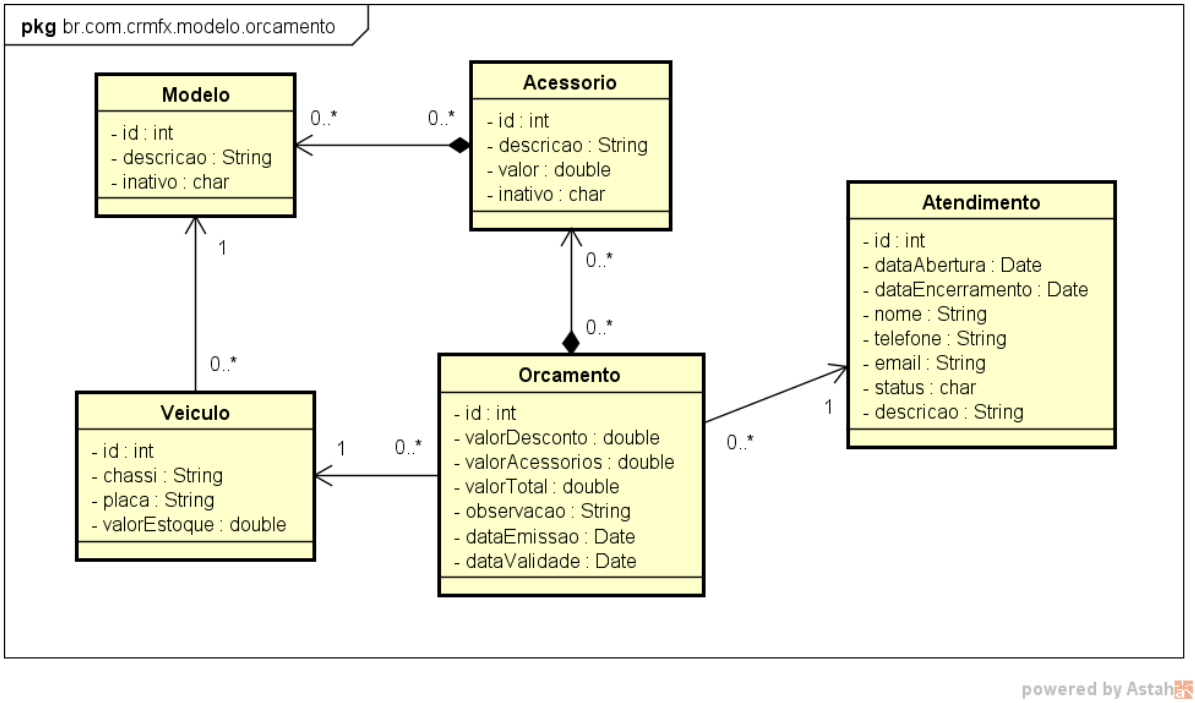
Os contatos realizados com o cliente durante um atendimentos são mantidos na classe Contato, ela possui relacionamento com as classes Usuario, Atendimento e LocalContato, onde: Um contato deverá ter um usuário, e um usuário poderá ter nenhum ou muitos contatos; Um contato deverá ter um atendimento, e um atendimento poderá ter nenhum ou muitos contatos; Um contato deverá ter um local de contato, e um local de contato poderá ter nenhum ou muitos contatos.

A classe VendaPerdida é responsável por manter as verbalizações dos clientes para os atendimentos encerrados com o status “P” (venda perdida). Possui relacionamento com a classe Atendimento, onde uma venda perdida deverá referenciar um atendimento, e um atendimento poderá ter nenhuma ou muitas vendas perdidas.

Todo atendimento deve pertencer a uma revenda, por isso a classe Revenda possui um relacionamento com a classe Atendimento, onde um atendimento deve referenciar uma revenda, e uma revenda pode ter nenhum ou muitos atendimentos.

Por fim a classe Pessoa possui um relacionamento com a classe Atendimento, a fim de identificar o cliente deste atendimento, onde um atendimento pode ter nenhum ou no máximo um ciente, e um cliente pode ter nenhum ou muitos atendimentos.

Figura 15: Diagrama de Classes, Módulo Atendimento



Fonte: Do autor, 2015

A classe Veiculo é responsável por armazenas os dados dos veículos em estoque, ela possui um relacionamento com a classe Modelo, onde um veículo deverá referenciar um modelo, e um modelo poderá ter nenhum ou muitos veículos.

Cada veículo poderá ter uma lista de acessórios disponíveis, conforme o seu modelo, com isso a classe Acessorio possui um relacionamento com a classe Modelo, onde um acessório poderá ter nenhum ou muitos modelos, e um modelo poderá ter nenhum ou muitos acessórios.

Já a classe Orcamento é responsável por manter os dados dos orçamentos emitidos, ela possui uma lista da classe Acessorio, onde um orçamento pode ter nenhum ou muitos acessórios, e um acessório pode ter nenhum ou muitos orçamentos. Ela também possui um relacionamento com a classe Atendimento, onde um orçamento deve ter um atendimento, e um atendimento pode ter nenhum ou muitos orçamentos.

3.7 AMBIENTE DE DESENVOLVIMENTO

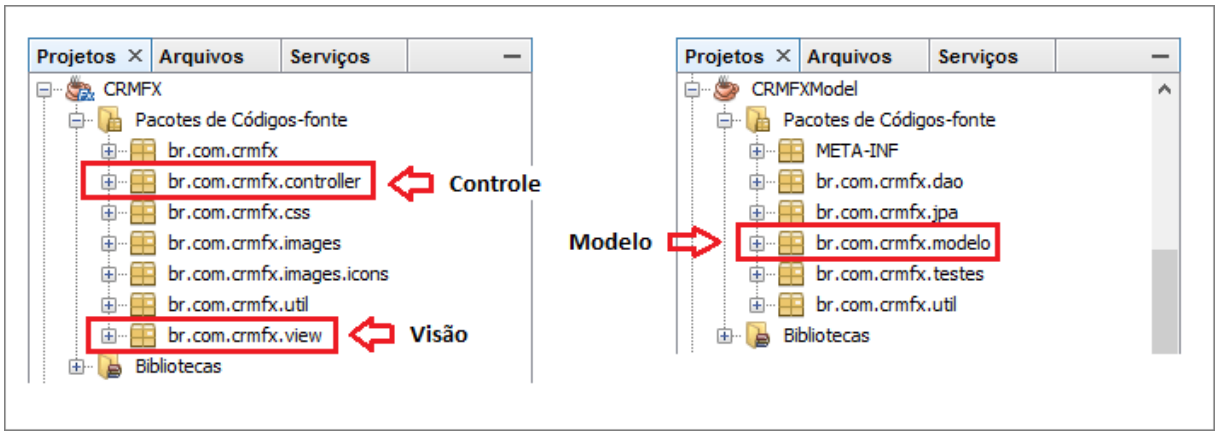
Para desenvolvimento do sistema foi utilizado o banco de dados PostgreSQL versão 9.1.0, NetBeans IDE 8.0.2 e JavaFX Scene Builder 2.0. O projeto seguiu o padrão MVC, neste capítulo é apresentada uma breve visão das camadas de Modelo, Visão e Controle e como as mesmas foram desenvolvidas.

3.7.1 O Modelo MVC

O desenvolvimento seguiu o padrão MVC, e como estratégia adotada, a camada de Modelo foi programada em um módulo separado das camadas de Visão e Controle, com isso, se caracteriza um fraco acoplamento entre elas, favorecendo assim, futuras manutenções, e até mesmo a utilização da camada de Modelo em outros projetos que necessitem conexão com este banco de dados.

A Figura 16 demonstra os projetos distintos das camadas de Modelo, Visão e Controle, onde o projeto que contém a camada de Modelo denomina-se CRMFXModel, e o projeto com as camadas de Visão e Controle denomina-se CRMFX.

Figura 16: Distribuição do modelo MVC no projeto



Fonte: Do autor, 2015

É possível observar que ambos os projetos possuem pacotes adicionais. Estes pacotes auxiliam e se integram aos pacotes de Modelo, Visão e Controle.

3.7.2 A Camada de Modelo

No projeto que contém a camada de modelo, é possível destacar três pacotes principais: *br.com.crmfx.modelo*, *br.com.crmfx.jpa* e *br.com.crmfx.dao*, além do pacote *META-INF* que contém o arquivo XML de configuração da JPA.

O pacote *br.com.crmfx.modelo* contém a declaração das Entidades da JPA, a Figura 17 mostra a declaração da Entidade LocalContato, seus atributos e anotações.

Figura 17: Declaração da Entidade LocalContato

```
@Entity
@Table(name = "local_contato")
public class LocalContato implements Serializable {
    @Id
    @Column(name = "id")
    @SequenceGenerator(name = "SEQ", sequenceName = "SEQ_LOCAL_CONTATO", allocationSize = 1)
    @GeneratedValue(generator = "SEQ", strategy = GenerationType.SEQUENCE)
    private Integer id;

    @NotEmpty(message = "A Descrição não pode ser nula")
    @Length(min = 5, max = 50, message = "A Descrição deve conter entre {min} e {max} caracteres")
    @Column(name = "descricao", length = 50, nullable = false)
    private String descricao;

    @Length(max = 1, message = "O Campo Inativo deve ser informado")
    @NotEmpty(message = "O Campo Inativo deve ser informado")
    @Column(name = "inativo", length = 1, nullable = false)
    private String inativo;

    // construtores, getters e setters
```

Fonte: Do autor, 2015

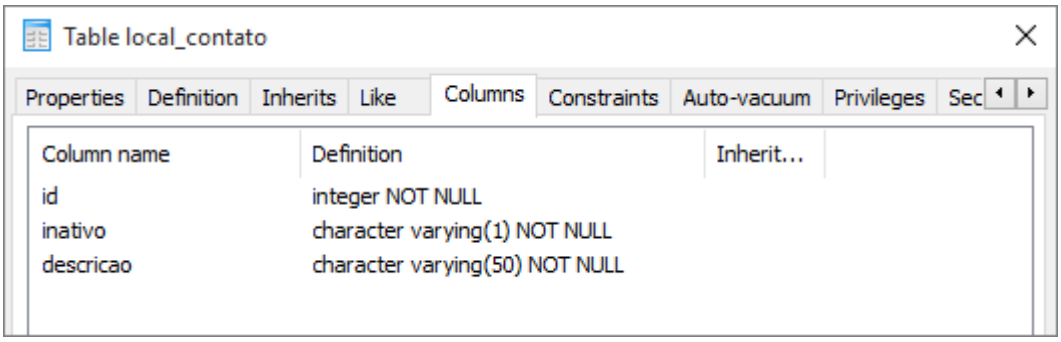
Para identificar a classe como uma entidade foi utilizado a anotação *@Entity*, e a anotação *@Table* indica o nome com o qual ela será representada no banco de dados. Para validação dos atributos foi utilizado as anotações do *Hibernate Validator*.

O atributo “id” possui a anotação @Id que define este atributo como chave primária da tabela no banco de dados, e a anotação *@Column* define o nome com o qual este atributo será representado no banco de dados. Os atributos *@SequenceGenerator* e *@GeneratedValue* indicam que será criada uma *sequence* no banco de dados para geração automática do valor deste atributo.

Os demais atributos possuem as anotações *@NotEmpty* e *@Length*, que definem respectivamente que o atributo não pode ser nulo, e o número mínimo e máximo de caracteres permitidos. Caso estas condições não sejam atendidas, é retornada uma mensagem personalizada por meio da propriedade *message*.

A Figura 18 demostra a representação da classe LocalContato no banco de dados PostgreSQL, seguindo as anotações anteriormente definidas.

Figura 18: Representação da classe LocalContato no PostgreSQL



Fonte: Do autor, 2015

O Pacote *META-INF* contém o arquivo XML de configuração exigido pela JPA, este arquivo é responsável pela comunicação da JPA com a base de dados. A Figura 19 mostra o arquivo de configuração *persistence.xml* do projeto.

Figura 19: Configuração da persistência, arquivo persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
                 http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
  <persistence-unit name="CRMFXModelPU" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <properties>
      <property name="javax.persistence.jdbc.url" value="jdbc:postgresql://server.local:5432/CRMFX"/>
      <property name="javax.persistence.jdbc.user" value="postgres"/>
      <property name="javax.persistence.jdbc.driver" value="org.postgresql.Driver"/>
      <property name="javax.persistence.jdbc.password" value="posttgres"/>
      <property name="hibernate.cache.provider_class" value="org.hibernate.cache.NoCacheProvider"/>
      <property name="hibernate.hbm2ddl.auto" value="update"/>
      <property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect"/>
      <property name="hibernate.connection.autocommit" value="false"/>
      <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.format_sql" value="true"/>
    </properties>
  </persistence-unit>
</persistence>
```

Fonte: Do autor, 2015

No arquivo persistence.xml é definido o nome da unidade de persistência como *CRMFXModelPU* no elemento *<persistence-unit>*, este nome identificará a unidade de persistência no projeto. No elemento *<provider>* foi definido o *Hibernate Persistence* como provedor da unidade de persistência.

O elemento com a propriedade < *property name="javax.persistence.jdbc.url"*> define as propriedades da conexão com o banco de dados, a propriedade <*property name="javax.persistence.jdbc.driver"*> define o driver JDBC do PostgreSQL a ser utilizado na conexão e a propriedade <*property name="hibernate.dialect"*> seta o dialeto padrão do SQL utilizado pelo *Hibernate*.

Foi desabilitada a funcionalidade de *auto commit*, alterando a propriedade <*property name="hibernate.connection.autocommit"*> para false, permitindo que seja efetuado de forma manual o *commit* de uma transação.

O pacote *br.com.crmfx.jpa* contem a classe EntityManagerUtil, que é responsável por instanciar a *Entity Manager Factory*, a Figura 20 mostra a classe EntityManagerUtil.

Figura 20: A classe EntityManagerUtil

```
package br.com.crmfx.jpa;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class EntityManagerUtil {

    public static EntityManagerFactory getEntityManagerFactory(){
        return Persistence.createEntityManagerFactory("CRMFXModelPU");
    }
}
```

Fonte: Do autor, 2015

O pacote *br.com.crmfx.dao* possui os *beans* responsáveis pela persistência dos objetos no banco de dados. Para se caracterizarem como um *bean* estas classes implementam a interface *serializable*, possuem um construtor nulo, e permitem acesso público aos seus métodos.

A Figura 21 mostra parte da classe LocalContatoDAO, no construtor foram instanciados as *Entity Managers*, e foi declarado o método *getEntityManager()* que retorna a *Entity Manager* instanciada.

Figura 21: A classe LocalContatoDAO

```
public class LocalContatoDAO implements Serializable {

    EntityManagerFactory emf;
    EntityManager em;

    public LocalContatoDAO() {
        emf = EntityManagerUtil.getEntityManagerFactory();
        em = emf.createEntityManager();
    }

    public EntityManager getEntityManager() {
        return em;
    }

    // metodos de persistência
```

Fonte: Do autor, 2015

A Figura 22 mostra o método *persistir()* da classe LocalContatoDAO, responsável por realizar a persistência do objeto LocalContato no banco de dados. Caso o objeto recebido possua um *id* nulo, é disparado o método *pesist()* da *Entity Manager*, caso contrário, é acionado o método *merge()*, para atualizar no banco de dados um objeto existente.

Figura 22: O método persistir da classe LocalContatoDAO

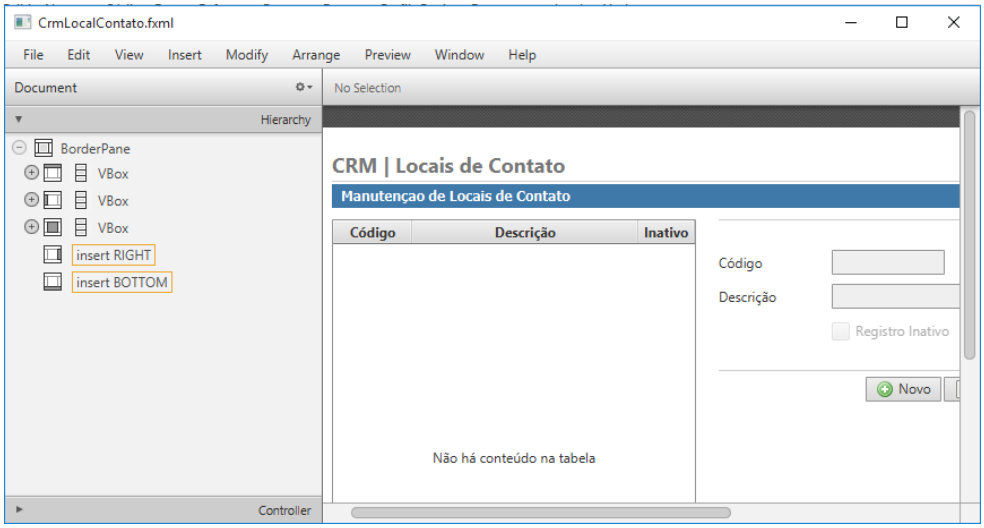
```
public void persistir(LocalContato objeto) throws Exception {
    EntityManager em = emf.createEntityManager();
    try {
        if (em.getTransaction().isActive() == false) {
            em.getTransaction().begin();
        }
        if (objeto.getId() == null) {
            em.persist(objeto);
        } else {
            em.merge(objeto);
        }
        em.getTransaction().commit();
    } catch (Exception e) {
        if (em.getTransaction().isActive() == false) {
            em.getTransaction().begin();
        }
        em.getTransaction().rollback();
        new Exception(e.getMessage());
    } finally {
        em.close();
        emf.close();
    }
}
```

Fonte: Do autor, 2015

3.7.3 A Camada de Visão e Controle

As telas do sistema que compõe a camada de visão foram desenvolvidas por meio da ferramenta JavaFX Scene Builder, na sua versão 2.0. A Figura 23: mostra a tela referente a manutenção dos locais de contato, no processo de desenvolvimento com a ferramenta JavaFX Scene Builder.

Figura 23: A tela Local de Contato no Scene Builder



Fonte: Do autor, 2015

As edições visuais de uma tela no Scene Builder são gravadas em um arquivo FXML, que será carregado em tempo de execução pelo JavaFX. A Figura 24 mostra o arquivo FXML da tela de manutenção de locais de contato.

Figura 24: Arquivo FXML da tela de manutenção de locais de contato

```
<BorderPane maxHeight="1.7976931348623157E308" maxWidth="1.7976931348623157E308" minHeight="500.0"
   <padding>
      <Insets bottom="20.0" left="20.0" right="20.0" top="20.0" />
   </padding>
   <top>
      <VBox BorderPane.alignment="CENTER">
         <children>
            <Label styleClass="lblHeaderWindow" text="CRM | Locais de Contato" />
            <Separator prefWidth="200.0" />
            <Label maxHeight="1.7976931348623157E308" maxWidth="1.7976931348623157E308" styleClass
               <VBox.margin>
                  <Insets bottom="10.0" />
               </VBox.margin>
            </Label>
         </children>
      </VBox>
   </top>
```

Fonte: Do autor, 2015

O JavaFX 8 suporta a estilização de seus componentes visuais por meio de comandos CSS (*Cascading Style Sheets*). O tema visual padrão do JavaFX 8 é chamado de Modena, que nada mais é do que um arquivo CSS, que pode ser encontrado compactado, no diretório de instalação do Java.

Para personalização dos componentes visuais, foi criado o arquivo *app.css*, localizado no pacote br.com.crmfx.css, a Figura 25 mostra parte do conteúdo deste arquivo, responsável pela estilização da *table view*.

Figura 25: Arquivo *app.css*

```
.table-view {
    -fx-border-radius: 1;
    -fx-background-radius: 1;
    -fx-background-insets: 1;
    -fx-border-color: #777777;
    -fx-padding: 0;
}

.table-view .column-header {
    -fx-size: 21;
    -fx-border-width: 0 0 1 0;
    -fx-border-color: #777777;
}
```

Fonte: Do autor, 2015

Os comandos CSS são praticamente os mesmos encontrados nas folhas de estilos de uma página web, com a exceção de que na grande maioria destes, deve conter o prefixo *-fx-.*

Para realizar alterações em um componente visual, como popular uma *combo box*, capturar o texto presente em um *text field* ou então programar o evento de um *button*, é necessário a vinculação de um *controller* à camada de visão FXML. A declaração do *controller* pode ser feita diretamente na guia Controller do JavaFX Scene Builder, ou então declarada diretamente no arquivo FXML, no componente de layout primário, como no exemplo: *<BorderPane minHeight="500.0" minWidth="800.0" fx:controller="br.com.crmfx.controller.LocalController">.*

A Figura 26 mostra parte do arquivo LocalController, que responsável pelo controle dos componentes da tela de manutenção de locais de contato.

Figura 26: Arquivo LocalController

```
public class LocalController implements Initializable {
    @FXML
    private TextField tfID, tfDescricao;
    @FXML
    private CheckBox cbInativo;
    @FXML
    private TableView<LocalContato> tbvLocal;
    @FXML
    private TableColumn<LocalContato, String> colID, colDescricao, colInativo;
    @FXML
    private Button btnNovo, btnEditar, btnSalvar, btnCancelar, btnExcluir;
    private LocalContatoDAO localContatoDAO;

    @FXML
    private void btnNovoAction(ActionEvent event) throws IOException {
        tbvLocal.getSelectionModel().clearSelection();
        clearForm();
        editando(true);
        Platform.runLater(() -> {
            tfDescricao.requestFocus();
        });
    }
```

Fonte: Do autor, 2015

Para que o controlador tenha acesso ao componente visual declarado no arquivo FXML, é necessário que o componente tenha informado o sua propriedade *fx:id*, que é o identificador do componente, e deverá ser único neste arquivo FXML. No exemplo: *<Button fx:id="btnNovo" onAction="#btnNovoAction" text="Novo">*, foi declarado um *fx:id* para o componente, e também uma ação a ser executada no momento em que foi clicado no botão.

Na Figura 26 é possível observar as anotações *@FXML*, que precedem algumas variáveis e métodos, isto indica que esta variável ou método esta declarado em algum componente no arquivo FXML.

3.7.4 Aplicação de Threads no Projeto

As telas da aplicação foram implementadas utilizando arquivos FXML, estes arquivos são carregados em tempo de execução, ou seja, no momento que o usuário realiza a abertura de determinada tela, é que a aplicação faz a leitura do arquivo FXML referente e monta seus componentes na tela. Juntando este modo de carregamento das telas com algumas operações no banco de dados, como por exemplo: na abertura da tela de manutenção de locais de contato, é realizada uma

busca no banco de dados pelos registros já cadastrados, caso seja encontrado um ou mais registros, estes são exibidos em uma *Table View*.

Estes processos acabam gerando um atraso na exibição da tela ao usuário, mesmo sendo um atraso pequeno, gera uma aparência de travamento da aplicação. Visando contornar este problema e apresentando uma melhor usabilidade usuário, foi utilizado o recurso de Threads, gerando fluxos paralelos para determinadas tarefas, como por exemplo: na abertura das telas de manutenção, é criada uma Thread para efetuar a busca pelos registros no banco de dados e iniciar os componentes da tela, como tamanho máximo de campos, mascaras e população de combo box.

A Figura 27 mostra a declaração e execução de uma Thread na inicialização da tela de manutenção de locais de contatos.

Figura 27: Declaração e execução de uma Thread

```
@Override
public void initialize(URL location, ResourceBundle resources) {
    SplashScreen.carregando("Carregando registros...");
    Task taskIniciaComp = new Task() {
        @Override
        protected String call() throws Exception {
            iniciaComponentes();
            return null;
        }
        @Override
        protected void succeeded() {
            SplashScreen.closeStage();
        }
        @Override
        protected void failed() {
            SplashScreen.closeStage();
            Util.janelaErro("Erro", null, "Erro ao iniciar componentes.");
        }
    };
    new Thread(taskIniciaComp).start();
}
```

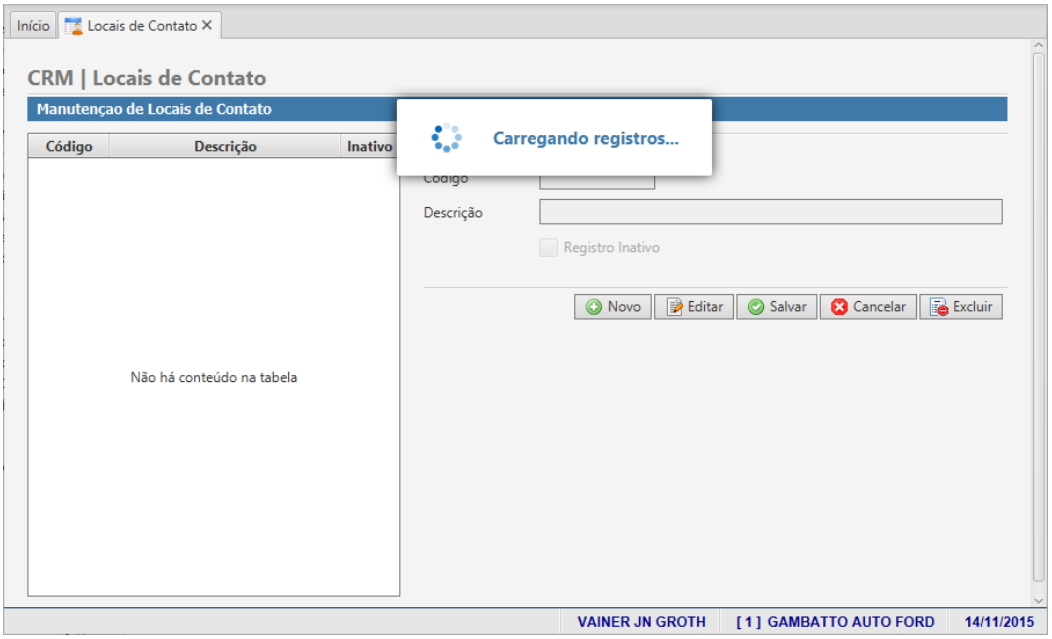
Fonte: Do autor, 2015

A inicialização da Thread é efetuada no método *initialize* do *controller*, nele é criada uma nova Task, onde é declarada as suas instruções de execução no método *call*(), em caso de erro na execução, é acionado o método *failed*(), que apresenta uma mensagem de erro.

Após a declaração da Task, ela é passada como parâmetro a uma nova Thread, que imediatamente é inicializada. É apresentada na tela uma mensagem para o usuário quando uma Thread estiver em execução, esta mensagem é modal,

para impedir que o usuário acione outra função da aplicação, ela será fechada quando a Thread finalizar suas operações. A Figura 28 mostra a tela de manutenção de locais de contatos no momento de sua inicialização, com a Thread sendo executada.

Figura 28: Thread em execução



Fonte: Do autor, 2015

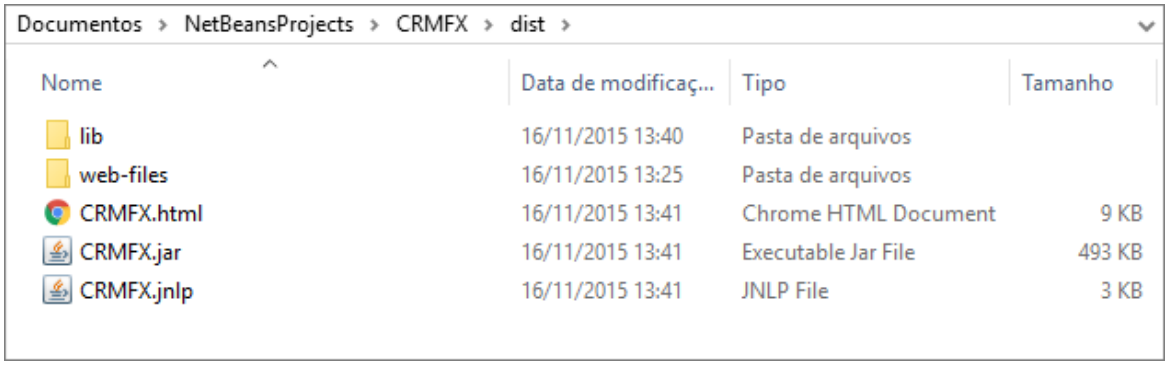
3.8 UTILIZAÇAO DA FERRAMENTA NO ESTUDO DE CASO

Uma aplicação JavaFX permite ser executada basicamente de três formas: Independência, como inicialização na web e no browser. A primeira se compara a uma aplicação desktop. A terceira faz uso da tecnologia Java web start, onde ela pose ser executada de forma remota sem o uso de um browser. Já a última é executada em um browser, como Google Chrome, Mozilla Firefox, entre outros.

Para demonstração do uso da aplicação, foi considerado o método de execução como independência, para os demais métodos é necessário a configuração do servidor da aplicação. No NetBeans a forma de execução é definida nas propriedades do projeto, ao gerar o *dist* da aplicação, é gerado uma estrutura com três arquivos de inicialização, onde o arquivo HTML será utilizado quando a aplicação roda por meio de um servidor web. Já o arquivo com extensão *.jar* será

utilizado quando a aplicação for executada localmente, como uma aplicação desktop, e por fim, quando a aplicação for executada utilizando o método Java Web Start, será utilizado o arquivo com extensão *.jnlp*. A Figura 29 mostra a estrutura do diretório *dist* da aplicação CRMFX.

Figura 29: Estrutura do diretório *dist* da aplicação



Fonte: Do autor, 2015

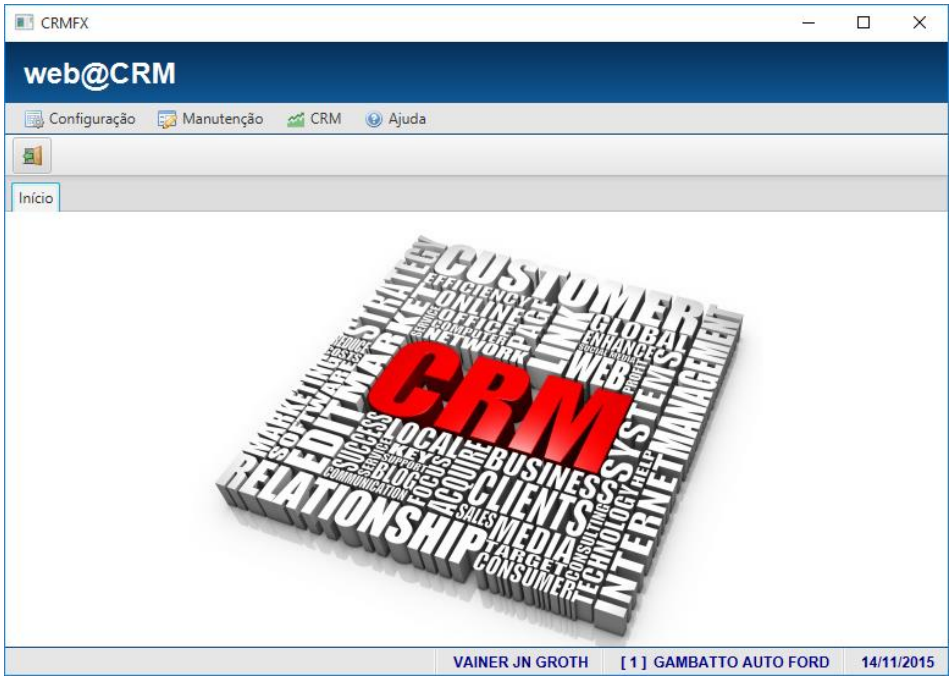
Ao executar a aplicação é apresentada uma tela de *login*, onde é necessário inserir um usuário e senha válidos para acesso. O campo data se refere à data na qual o sistema irá se basear na abertura ou encerramento de um atendimento, registro de um contato ou emissão de um orçamento. Por padrão será carregado a data atual do sistema operacional do usuário. A Figura 30 mostra a tela de *login* da aplicação.

Figura 30: Tela de *login*

Fonte: Do autor, 2015

Após inserir um usuário e senha válidos, é carregada a tela principal da aplicação. A tela possui na parte superiora um menu, para acesso as suas funcionalidades, logo abaixo há uma barra de ferramentas, com atalhos para as principais áreas da aplicação. Na parte central da tela há um painel de abas, onde cada tela da aplicação será uma nova aba deste painel. A Figura 31 mostra a tela principal da aplicação.

Figura 31: Tela principal

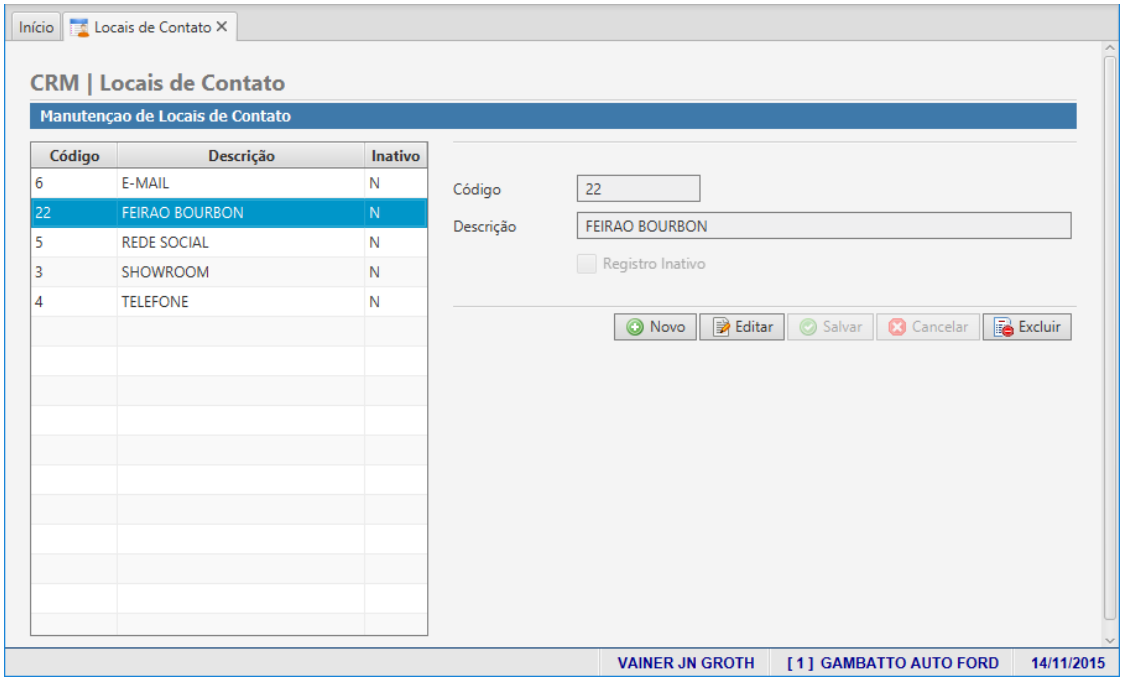


Fonte: Do autor, 2015

No rodapé da tela é mostrado o nome do usuário que efetuou *login* na aplicação, o código e o nome da revenda padrão deste usuário, e a data informada ao efetuar o *login*.

As telas da aplicação seguem um padrão, quando possível. A Figura 32 mostra a tela de manutenção dos locais de contato, as demais telas do sistema respeitam este padrão.

Figura 32: Tela de manutenção de locais de contato

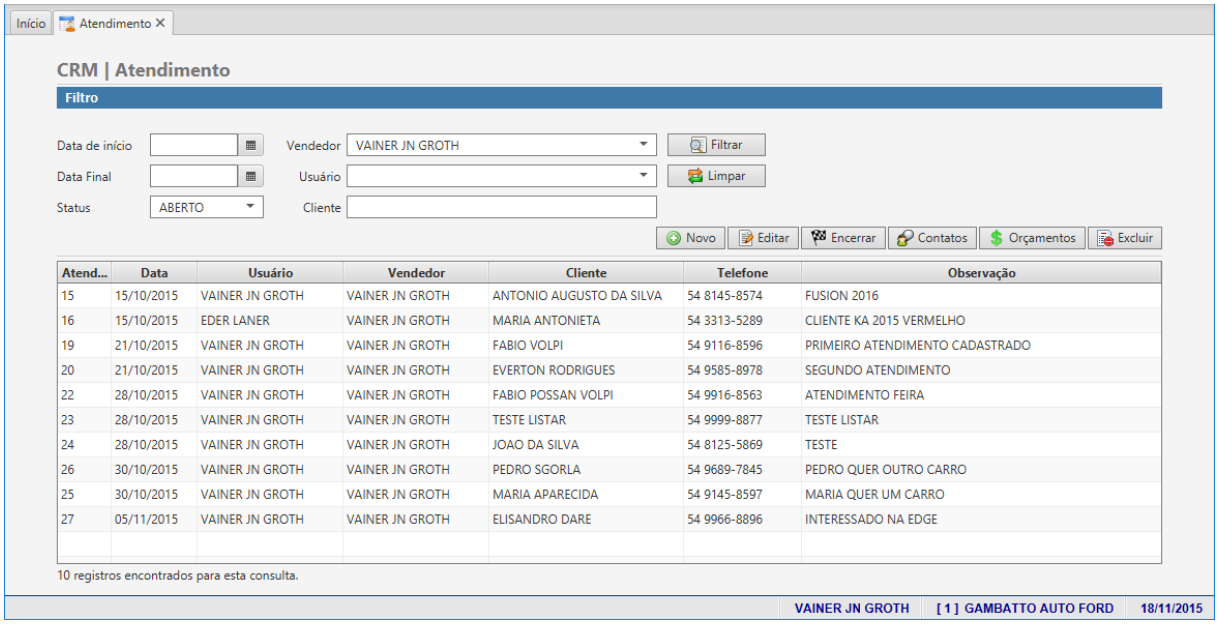


Fonte: Do autor, 2015

A lateral esquerda da tela contém uma lista dos registros, onde é possível selecionar um registro para visualização ou edição. Já ao lado desta lista esta o formulário com os campos que compões o cadastro do registro, e, por fim, abaixo deste encontra-se os botões de controle, para cadastro de um novo registro (botão novo), edição de um registro selecionado (botão editar), salvar um registro (botão salvar), cancelar uma alteração (botão cancelar) e excluir um registro selecionado (botão excluir).

O fluxo principal do sistema ocorre na manutenção dos atendimentos e registro dos contatos realizados. A Figura 33 mostra a tela de gerenciamento de atendimentos, é nesta tela que um usuário tem acesso aos atendimentos realizados pela equipe de vendas. Na abertura da tela é verificado se o usuário logado na aplicação também é um vendedor, se este for, é realizada uma busca de forma automática, por todos os atendimentos que estiverem com o status “Aberto” para este vendedor.

Figura 33: Tela de Atendimento

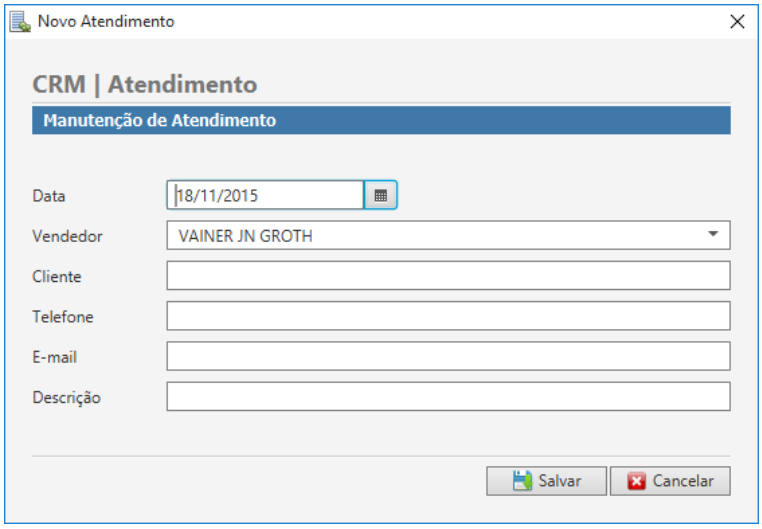


Fonte: Do autor, 2015

Na parte superior da tela, a aplicação traz um formulário, onde permite que um usuário possa realizar uma busca de forma personalizada por atendimentos existentes. Ao informar os campos desejados e clicar no botão Filtrar, o sistema busca imediatamente por atendimentos que atendam as condições definidas pelo usuário.

Para cadastrar um novo atendimento, o usuário deve clicar no botão Novo, a Figura 34 mostra a tela de inserção de um novo atendimento.

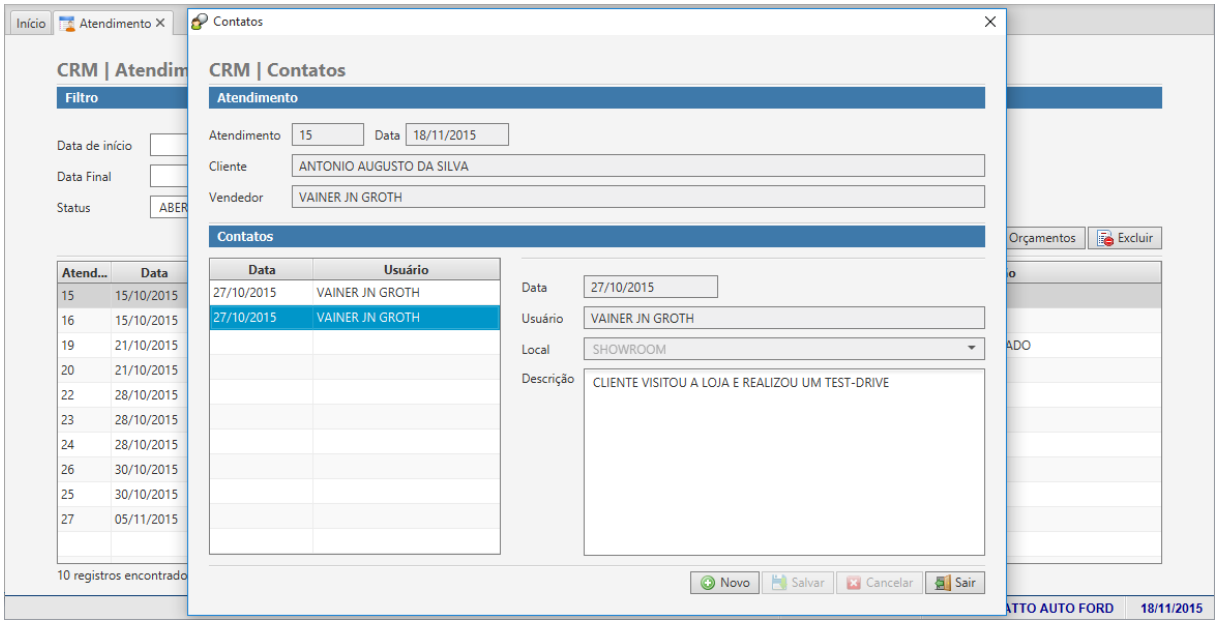
Figura 34: Novo Atendimento



Fonte: Do autor, 2015

Para um atendimento já cadastrado, e com status “Aberto” é possível registrar os contatos realizados com o cliente para este atendimento. A Figura 35 mostra a tela de gerenciamento de contatos, nela é possível verificar os contatos já informados e informar novos contatos.

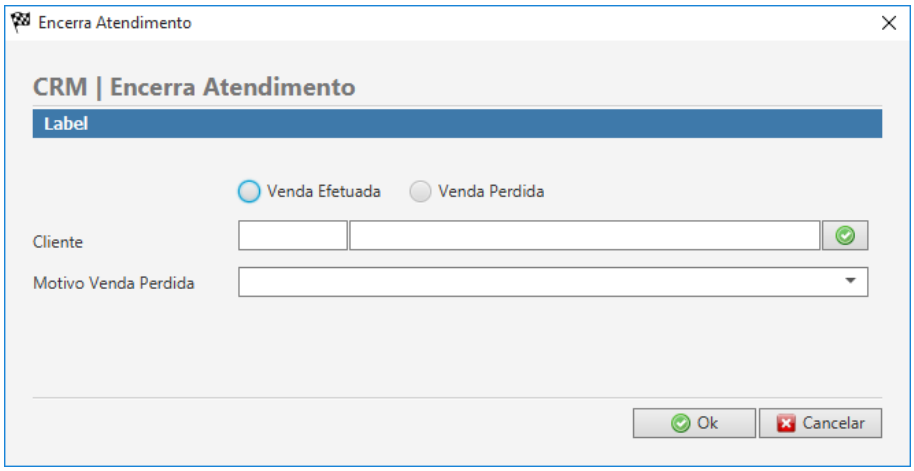
Figura 35: Contatos de um Atendimento



Fonte: Do autor, 2015

Para encera um atendimento, o usuário deve informar o cliente do atendimento, em caso de venda efetuada, ou o motivo da perda, em caso de venda não efetuada. A Figura 36 mostra a tela de encerramento de um atendimento.

Figura 36: Tela de encerramento de um Atendimento



Fonte: Do autor, 2015

4 CONSIDERAÇÕES FINAIS

Considerando a concorrência atual do mercado, o desenvolvimento de um sistema, com foco na resolução de necessidades específicas, pode ser considerado peça chave na aceleração da tomada de decisões, e na organização da informação.

O presente estudo teve como objetivo compreender o uso de interfaces RIA, mais especificamente JavaFX, juntamente com os conceitos de orientação a objeto e padrões MVC. Também abordou o uso de JPA com provedor de persistência Hibernate, e por fim o uso destas tecnologias para desenvolvimento de um sistema CRM para uma concessionária de veículos, com base em um estudo de caso.

Para o desenvolvimento do sistema foi necessário, além de retomar o estudo sobre conceitos de orientação a objetos e JPA, também conhecer as particularidades do JavaFX, o que resultou em uma maior curva de aprendizado. A principal dificuldade encontrada foi referente ao levantamento do referencial teórico sobre JavaFX, que no inicio, se resumia a pequenos exemplos encontrados na internet. A programação da aplicação também sofreu atrasos, oriundos da necessidade de adquirir conhecimento sobre o funcionamento do JavaFX, para poder colocá-los em prática. Na medida em que a aplicação se desenvolvia a programação se equilibrou.

JavaFX se mostrou prático e eficiente na implementação da aplicação, a construção das telas por meio da ferramenta JavaFX Scene Builder foram realizadas de forma descomplicada, e a personalização dos componentes foram aplicadas facilmente por meio de arquivos com instruções CSS , o que pode exemplificar a praticidade na utilização dos recursos de JavaFX no desenvolvimento de aplicações RIA.

A versão atual do sistema não está finalizada, mas os módulos atuais permitem a manutenção de cidades e pessoas, bem como revendas e usuários, este ainda necessita de ajustes relativos a permissões. No módulo CRM é possível realizar a manutenção de locais de contatos, motivos de venda perdida e modelos e acessórios de veículos, por fim permite a manutenção de atendimentos, bem como os contatos realizados.

Mesmo na versão atual do sistema imagina-se que há uma melhora na organização da informação, pois permite um fácil acesso aos detalhes de uma negociação realizada pela revenda. Como trabalhos futuros restam a implementação

do módulo de orçamentos, ajustes referentes as permissões de usuários e a emissão de relatórios para acompanhamento dos processos realizados. É possível também a implementação do recurso de envio de e-mail para o gerente, vendedor ou até mesmo o cliente, com detalhes da negociação realizada, visando obter uma melhor interação entre usuário e sistema.

REFERÊNCIAS

AGOSTINI, Cristiano; RODRIGUES, Daniel. **Construindo aplicações de interface rica com JavaFX**. Unoesc & Ciência-ACET, 2010. Disponível em: <http://editora.unoesc.edu.br/index.php/acet/article/viewFile/157/pdf_77> Acesso em: set. 2014.

BRINGHENTI, Fábio. **Análise e desenvolvimento de um Sistema Web de Gestão para Clínicas de Saúde**. 2015. Monografia (Graduação em Tecnólogo em Sistemas para Internet) – Instituto Federal Sul-Rio-Grandense, Passo Fundo, 2015. Disponível em: <http://inf.passofundo.ifsul.edu.br/uploads/arq/20150716093526734444460.pdf> Acesso em: set. 2015.

CHAPPELL, G.; POTTS, J.; HILDEBRANDT, N. **JavaFX: Getting Started with JavaFX**, Release 2.2.40. ORACLE. 2013. Disponível em: <http://docs.oracle.com/javafx/2/get_started/jfxpub-get_started.pdf> Acesso em: out. 2014.

DE MEDEIROS, Ernani Sales. **Desenvolvendo Software com UML 2.0: definitivo**. São Paulo: Pearson Makron Books, 2004.

DEA, Carl; COFFIN, David. **JavaFX 2.0: introduction by example**. [S.I.]: Apress, 2011.

_____ et al. **JavaFX 8: Introduction by Example**. [S.I.]: Apress, 2014.

DEITEL, Harvey M; DEITEL, Paul J. **Java: Como Programar**. São Paulo: Pearson Prentice Hall, 2010.

FEDORTSOVA, I. JavaFX: Mastering FXML, Release 2.2. ORACLE. 2013. Disponível em: <http://docs.oracle.com/javafx/2/fxml_get_started/jfxpub-fxml_get_started.pdf > Acesso em: set. 2014.

GREENBERG, Paul. **CRM, Customer Relationship Management na velocidade da luz: conquista e lealdade de clientes em tempo real na internet**. Rio de Janeiro: Campus, 2001.

MARQUEZINI, Alex de Souza; AGUINALDO, Emerson Ceruti; ALMEIDA, Osvaldo Cesar Pinheiro. **Desenvolvimento de Aplicação de ponto de venda usando JavaFX**. Botucatu: Tekhne e Logos, 2013. Disponível em: <http://www.fatecbt.edu.br/seer/index.php/tl/article/download/242/191 > Acesso em: Out 2014.

OLIVEIRA, Bruno. **JavaFX: interfaces com qualidade para aplicações desktop**. São Paulo: Casa do Código, 2013.

PINA, DSA; OLIVEIRA, LEMC. **RIA-Rich Internet Applications: Uma Revisão dos principais expoentes da área.** Disponível em: <http://www.unibratec.edu.br/tecnologus/wp-content/uploads/2013/10/tecnologus_edicao_07 _artigo_05.pdf> Acesso em: Set. 2013.

TOPLEY, Kim. **JavaFX developer's guide**. [S.I.]: Pearson Education, 2011.

WEAVER, J. L.; GAO, W.; CHIN, S.; IVERSON, D.; VOS, J. **Pro JavaFX 2: A Definitive Guide to Rich Clients with Java Technology**. New York: Apress, 2012.

ANEXOS

ANEXO A - Planilha de controle das visitas de clientes à concessionária.

Gambatto Auto ltda.

CONTROLE DE FLUXO DE CLIENTES

DATA: ___01___/___11_____/___2014_____			MOTIVO DA VISITA				
CLIENTE	TELEFONE	VENDEDOR	1ª VISITA	RETORNO/ EM NEGOC.	ENTREGA DOC.	RETIRADA VEÍCULO	DIVERSOS
GILBERTO RENNER	[redacted]	DILAMAR	X				NOVO
JULIO CESAR PAVIN	[redacted]	LUCIANE	X				NOVO
ADRIANO DA SILVA	[redacted]	DIENIFER	X				NOVO
TIAGO DOS SANTOS	[redacted]	ALEX		X			
CASSIANO ROCHA	[redacted]	CLIDEMAR	X				NOVO
DIAN BORGES	[redacted]	WILSON	X				CONSÓRCIO
EUDES CAZARIN	[redacted]	TIAGO	X				NOVO
JOÃO PINHEIRO	[redacted]	WILSON	X				SEMI NOVO
DIONEI WOMMER	[redacted]	ROGÉRIO	X				SEMI NOVO
ELIZETE LOURENCI	[redacted]	TADEU	X				NOVO
LUCIANO ALMEIDA	[redacted]	ALEX		X			
VILMAR VIEIRA	[redacted]	DILAMAR	X				NOVO
AISLON FREITAS	[redacted]	WILSON		X			
HENRIQUE HABIB	[redacted]	CLIDEMAR	X				NOVO
TALES BARBOSA	[redacted]	ALEX	X				NOVO

ANEXO B - Ficha de Negociação preenchida pelo vendedor.

Ford Gambatto

TERMO DE NEGOCIAÇÃO

( ) ESTOQUE ( ) FROTISTA ( ) TAXI ( ) PORTADOR DE DEFICIÊNCIA
( ) PRODUTOR RURAL ( )INTERNET

DADOS DO FATURAMENTO DATA DA VENDA:_____

VEÍCULO:_____CAT:_____CHASSI:_____PLACA:_____COR:_____

NOME:_____ DATA NASC:_____

RG:_____CPF:_____E MAIL:_____

RUA:_____N°:_____BAIRRO: _____COMPL:_____

CEP:_____CIDADE:_____FONE:_____

ESTADO CIVIL_____ PROFISSÃO_____

( )LIBERADO ( )LEASING ( )CDC ( )PROGER ( )CONSÓRCIO
ALIENAÇÃO:_____

CNPJ DO BANCO: _____VENDEDOR:_____VL COMISSÃO:_____

DADOS RECEBIMENTO DO USADO:

VEÍCULO:_____ANO/MOD:_____PLACA_____VALOR:_____

REPASSADO PARA:_____VALOR:_____

AUTORIZADO POR: _____ VISTO: _____

RESUMO DO NEGÓCIO:

Valor do Estoque:_____ BANCO: _____

Valor Desconto:_____ VALOR PARC.:_____Ass:_____

Valor de Acessórios:_____ PRAZO:_____

Total da Venda:_____ COEFICIENTE:_____

Valor Financiado:_____ 1°VENC.:_____

Veículo Usado:_____ TAC:_____

A Vista: _____ VALOR DA NF _____

OBS: _____

DECLARO QUE ESTOU PLENAMENTE CIENTE DOS TERMOS AVENÇADOS NA PRESENTE NEGOCIAÇÃO.

_____ _____ _____
CIENTE CLIENTE VENDEDOR VISTO GERÊNCIA

VISTO FINANCEIRO_____

APROVAÇÃO R$:_____ VISTO DO OPERADOR:_____