



# Universidade Norte do Paraná

---

SISTEMA DE ENSINO PRESENCIAL CONECTADO  
TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MARCELO EDUARDO ALVES PAIXÃO RESENDE

## **SISTEMA DE PET SHOP**

MARCELO EDUARDO ALVES PAIXÃO RESENDE

## **SISTEMA DE PET SHOP**

Trabalho em grupo apresentado à Universidade Norte do Paraná - UNOPAR, como requisito parcial para a obtenção de média semestral nas seguintes disciplinas: Linguagens de Programação e Estrutura de Dados, Banco de Dados I, Organização de Computadores e Análise Orientada a Objetos I.

Orientador(es):

Prof. Anderson Emídio de Macedo Gonçalves;

Prof.<sup>a</sup> Merris Mozer;

Prof.<sup>a</sup> Iolanda Claudia Sanches Catarino

Prof. Leonardo de Marchi Ferrareto

Prof. Paulo Kiyoshi Nishitani

Uberlândia  
2016

## SUMÁRIO

1	INTRODUÇÃO.....	3
2	OBJETIVO.....	4
2.1	OBJETIVO GERAL.....	4
2.2	Objetivos específicos.....	4
3	DESENVOLVIMENTO.....	5
3.1	ESCOPO.....	6
3.2	Caso de estudo: Empresa Cats & Dogs.....	7
3.3	Engenharia de Software e UML.....	8
3.4	Diagrama de caso de uso.....	9
3.5	Diagrama de classes.....	11
3.5.1	Pacote Serviço.....	13
3.5.2	Pacote Acesso.....	13
3.5.3	Pacote Animal.....	13
3.5.4	Pacote Cliente.....	13
3.5.5	Pacote Agendamento.....	14
3.5.6	Pacote Relatórios.....	14
3.6	Programação e Estruturas de Dados da Aplicação.....	15
3.6.1	Estrutura de dados e Regras de Negócio.....	15
3.6.1.1	Fila.....	15
3.6.1.2	Fila Priorizada.....	16
3.6.2	Classe Agenda como uma estrutura complexa.....	16
3.7	Modelo de Banco de Dados.....	20
3.7.1	Modelo Conceitual.....	20
3.7.2	Modelo Lógico.....	21
3.8	Microcomputadores e processos de negócio.....	22
3.8.1	Vantagens.....	22
3.8.1.1	Concorrência.....	22
3.8.1.2	Proximidade do cliente.....	22
3.8.2	Desvantagens.....	23
3.8.2.1	Segurança.....	23
3.8.2.2	Custo inicialmente alto.....	23

3.9	Ambiente e Hardware.....	24
4	Conclusão.....	25
	REFERÊNCIAS.....	26

## 1 INTRODUÇÃO

O presente trabalho é um estudo de caso da elaboração e desenvolvimento de um sistema informatizado a ser implantado, a fim de melhorar os processos, em um negócio já existente. As informações obtidas e produzidas durante este trabalho servirão para enriquecimento do conhecimento acerca dos processos de análise e desenvolvimento de softwares de *PetShops*, das suas necessidades, do entendimento das suas regras de negócio e dos resultados que podem ser obtidos.

## 2 OBJETIVO

### 2.1 OBJETIVO GERAL

Para atender o objetivo geral deste trabalho, pretende-se realizar um levantamento das informações disponíveis sobre como os negócios de PetShops operam, de uma forma geral, e como funcionam seus processos de negócio e suas regras de negócio. Para atingir este objetivo, serão utilizadas informações disponíveis na internet e casos de estudo fornecidos.

### 2.2 OBJETIVOS ESPECÍFICOS

A fim de alcançar o objetivo geral, os seguintes objetivos específicos serão seguidos:

- Criar uma análise de dados, identificando requisitos necessários para o desenvolvimento do software;
- A partir dos requisitos identificados, diagramas serão criados utilizando a linguagem UML, para que assim toda a lógica, arquitetura e dinâmica do software seja de fácil entendimento;
- Criar o diagrama de caso de uso utilizando os elementos fornecidos pela UML;
- Criar o diagrama de classes, para auxiliar e registrar os dados obtidos na análise orientada a objetos;
- Criar a modelagem do banco de dados, de forma genérica e reutilizável, através do Diagrama de Entidade e Relacionamento;
- Exemplificar de forma prática e facilmente reproduzível como deve ser guiada a lógica de um software a ser desenvolvido, a fim de validar os dados identificados contra as regras de negócio, e para isso utilizando uma linguagem de programação;
- Identificar as vantagens e desvantagens da utilização de uma plataforma baseada em microcomputadores para o desenvolvimento deste tipo de software.

### **3 DESENVOLVIMENTO**

Para o desenvolvimento deste estudo, cada etapa será descrita de forma incremental. Assim como no desenvolvimento do software em si, cada etapa será detalhada de forma que possa ser analisada incrementalmente, com cada etapa se construindo em cima dos dados da anterior.

### 3.1 ESCOPO

Este software controlará o agendamento de serviços fornecidos por um Pet Shop, mantendo dados e estado de forma fidedigna dos clientes, funcionários e animais. Deverá permitir que funcionários, de qualquer nível hierárquico da empresa, possam obter, alterar ou remover dados, de acordo com seu nível de permissão, levando em conta concorrência de utilização e restrições de tempo.

O software deve ser capaz de minimizar o tempo gasto nas tarefas administrativas e de processamento de dados de clientes, para que assim os funcionários possam maximizar seu desempenho e tempo disponível nas atividades e negócio da empresa, sem no entanto comprometer a confiabilidade e rigidez destes dados. Para isto, o software deve se encarregar de validar todos os dados de entrada e saída, apresentando os erros e avisos de forma clara e objetiva.



### 3.2 CASO DE ESTUDO: EMPRESA CATS & DOGS

Para melhor entendimento do conceito deste estudo, será utilizado o exemplo de uma empresa fictícia, chamada Cats & Dogs, onde o desenvolvimento do software poderia ser feito e implementado no negócio desta. Sendo assim, a proposta deste trabalho deve atender com precisão as necessidades do caso de estudo.

A empresa Cats & Dogs trabalha no ramo de PetShop, é de porte pequeno e teve um aumento significativo no número de clientes e serviços executados. Com este aumento de serviços prestados, surgiu um problema na administração dos dados produzidos decorrentes da atividade de negócio da empresa, onde o controle dos agendamentos e serviços prestados ficou cada vez mais complexo e impreciso, sendo até então feito de forma manual.

Apesar de atuar também na venda de produtos, os requisitos mais urgentes explicitam a necessidade de automatizar primeiramente os registros de agendamentos e serviços prestados. Para um maior reaproveitamento do esforço de desenvolvimento, a construção deve levar em conta a extensibilidade, permitindo que futuramente novas funcionalidades, como a possibilidade de automatizar a venda de produtos, seja adicionada sem trazer maiores custos.

### 3.3 ENGENHARIA DE SOFTWARE E UML

Durante as etapas de Engenharia de Software e Análise de Sistemas, um grande volume de dados são identificados e é necessário que uma documentação seja produzida, a fim de permitir uma melhor comunicação com os envolvidos no projeto no momento e com todos os outros profissionais que possam vir a se envolver com este.

Para esta finalidade, a linguagem de modelagem UML (Unified Modeling Language) será a opção escolhida neste estudo, por fornecer um grande número de elementos gráficos e textuais utilizados em projetos no mundo todo, e portanto, os diagramas que serão produzidos neste estudo poderão ser compreendidos e estudados por um grande número de profissionais.

De acordo com MACORATTI (2004), “A UML - Unified Modeling Language - é um modelo de linguagem para modelagem de sistemas orientados a objetos (principalmente) . Com ela podemos fazer uma modelagem visual de maneira que os relacionamentos entre os componentes do sistema sejam melhor visualizados e compreendidos e documentados.

Podemos dizer também que a UML é uma linguagem para especificar , construir , visualizar e documentar um sistema de software [...]”.

### 3.4 DIAGRAMA DE CASO DE USO

Para representar o comportamento do sistema, os objetivos dos usuários ao utilizá-lo e guiar o desenvolvimento das funcionalidades, a UML nos permite criar Diagramas de Casos de Uso que representam com elementos claros e simples como e quem utiliza o sistema.

Segundo LARMAN (2011, p. 36), “A análise de requisitos pode incluir narrativas ou cenário sobre como as pessoas usam a aplicação; estes podem ser escritos como casos de uso. Casos de uso não são artefatos orientados a objetos - eles são simplesmente narrativas escritas. Contudo, são uma ferramenta popular para utilização na análise de requisitos. [...]”.

Antes da criação dos diagramas em si, uma simples enumeração do que será representado graficamente é necessária.

Dos atores que utilizarão o sistema:

- Cliente: um ator que não interage administrativamente com o sistema, mas dá início a um caso de uso de agendamento;
- Atendente: é qualquer funcionário que esteja executando a função de atendimento e tem permissão de administrar dados relativos ao serviço prestado;
- Gerente: representa um funcionário com permissão total sob o sistema e que exerce atividades gerenciais, como criação, exclusão e alteração de usuários e também a emissão de relatórios.

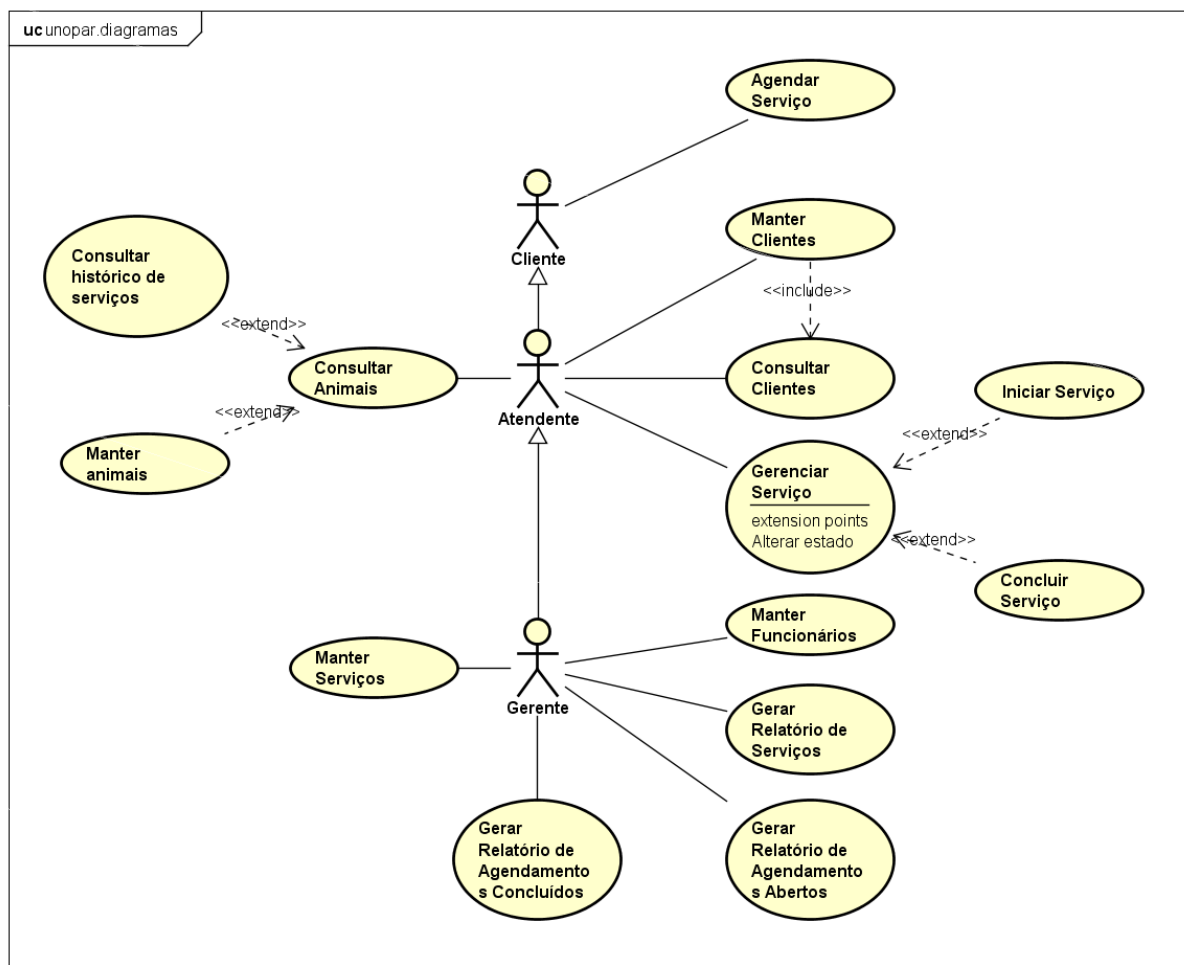
Dos casos de uso e histórias de usuário:

- O software DEVE manter nome, telefone e endereço de um Cliente;
- O software DEVE fornecer funcionalidade de pesquisa de Clientes pelos dados previamente cadastrados e exibir seus dados, caso encontre resultado;
- O software DEVE sugerir o cadastramento de um cliente caso a pesquisa de Clientes não encontre resultados;
- O software DEVE manter nome, tipo (cão, gato...), raça, sexo e cliente dono de um Animal;
- O software DEVE fornecer a possibilidade de imprimir para o usuário um histórico dos serviços realizados;
- O software DEVE fornecer, com a devida autorização, a possibilidade de adicionar, alterar e remover funcionários (usuários);
- O software DEVE fornecer, com a devida autorização, a possibilidade de adicionar, alterar a remover serviços;
- O software DEVE fornecer a possibilidade do funcionário ou cliente criarem um registro de agendamento;
- O software DEVE manter o estado de cada agendamento e alterar o estado para refletir a situação do serviço que está sendo realizado;

- O software DEVE fornecer a possibilidade de um funcionário, com permissões gerenciais, emitir relatórios dos agendamentos do dia, escolhendo entre agendados e concluídos, e também dos clientes, animais e serviços realizados.

Atendendo a todos os dados obtidos sobre o negócio, um diagrama de caso de uso foi produzido.

Figura 1 – Diagrama de caso de uso



powered by Astah

Fonte: Produzido pelo pesquisador.

### 3.5 DIAGRAMA DE CLASSES

O diagrama de classe é um dos modelos gráficos do Modelo Unificado de Linguagem (Unified Modeling Language - UML) e representa como as classes do sistemas estão estruturadas, como se relacionam, e como colaboram entre si para representar objetos do mundo real, apresentando uma visão estática de um projeto orientado a objetos. É fundamental no desenvolvimento de sistemas e serve de base para a construção de outros diagramas, como o de Comunicação, de Sequência e Estados.

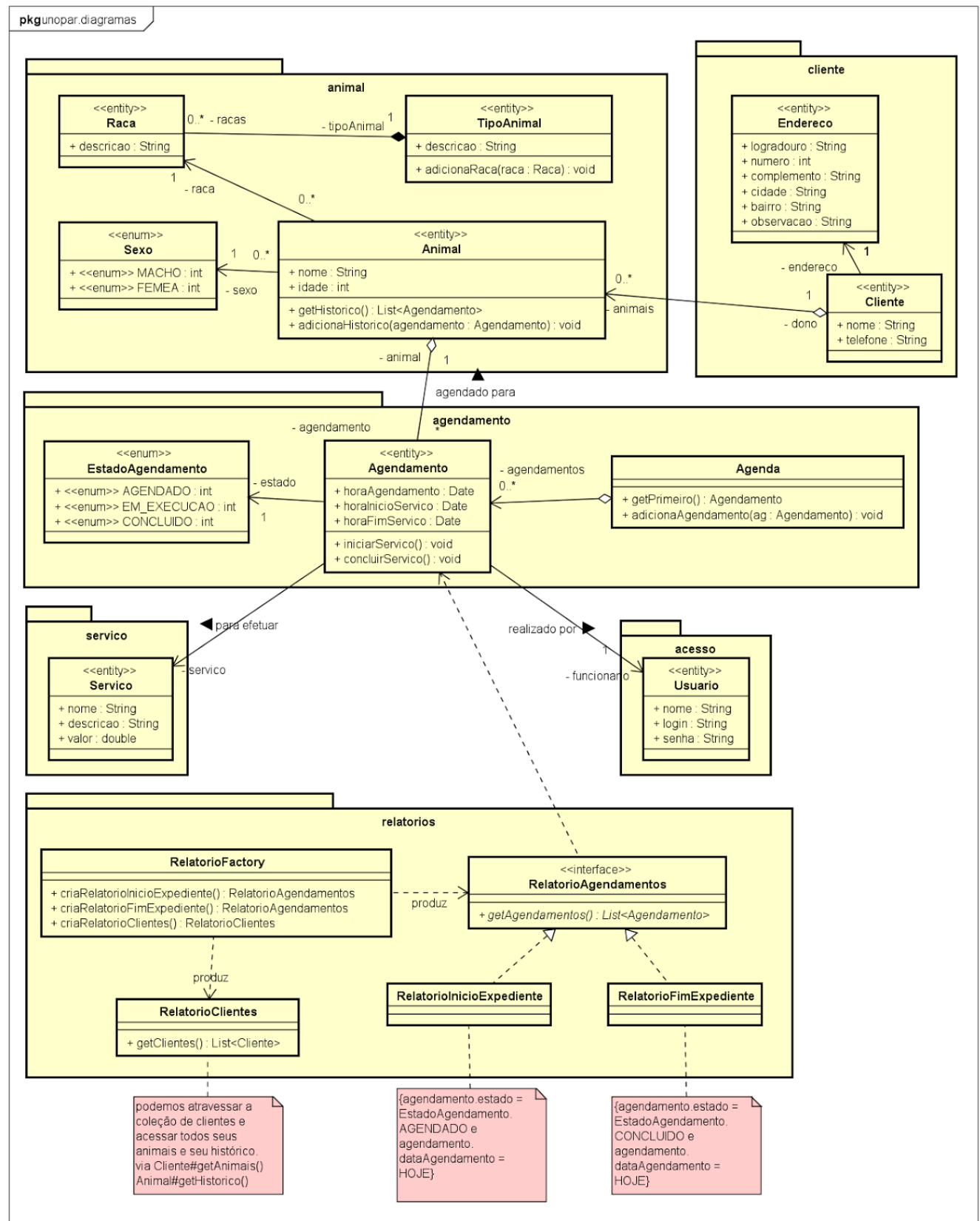
Para o diagrama de classes deste estudo, por ser conceitual, detalhes estritamente técnicos como relacionamento com tecnologias específicas ou interação com uma camada de persistência e de apresentação não foram representados. Também, outras observações devem ser levadas em consideração para melhor entendimento.

Cada classe tem seus atributos específicos e estes foram precisamente representados no diagrama, porém a forma de acesso a eles foi omitida, já que é dependente da linguagem que será escolhida para implementação e não adiciona informação significativa para o entendimento do mesmo.

Atributos que serão inevitavelmente necessários durante a implementação, mas que não auxiliam no entendimento do negócio, foram também omitidos, como atributos de código das entidades.

Métodos que salvam, alteram e apagam registros de entidades não foram representados no diagrama, pois também dependem de implementação. Entretanto, foi utilizado o elemento de estereótipos, fornecido pela UML e parte da especificação desta, para marcar classes que representam entidades, e uma implementação final deste modelo deve permitir a recuperação, inserção, remoção e alteração destas. Após a observação de todos estes detalhes, o diagrama produzido pode ser conferido na imagem a seguir

Figura 2 – Diagrama classes



Como pode ser visto na figura, foi adotada uma abordagem mais rica em quantidade de elementos utilizados para representar características da organização, funcionalidade e regras do sistema, como o uso de pacotes e estereótipos. Cada um dos pacotes será detalhado a seguir.

#### 3.5.1 Pacote Serviço

O pacote serviço contém classes que representem ou operem sobre os serviços prestados pela empresa. Na versão apresentada neste estudo, o pacote só contém uma classe de entidade também chamada de Serviço, que representa um serviço prestado pela empresa, contendo nome e valor. Da forma como foi modelado, permite que os funcionários da empresa, com devida permissão, alterem e criem novos serviços de acordo com o que a empresa estiver oferecendo, sem que seja necessária uma manutenção do código do sistema.

#### 3.5.2 Pacote Acesso

O pacote acesso contém todas as classes que tratem da segurança da aplicação, ou seja, permissão de uso e restrição. Uma única classe está contida neste pacote, que é a entidade Usuário.

#### 3.5.3 Pacote Animal

O pacote de animal contém a classe de entidade Animal, uma das principais do sistema, e as demais classes e entidades que colaboram entre si para representar a entidade principal de forma modular. A classe Animal também contém métodos específicos que permitem a recuperação do histórico de serviços realizados e adição destes, vinculando-os ao animal.

#### 3.5.4 Pacote Cliente

O pacote cliente armazena duas entidades: Endereço e Cliente. Todas classes que representem e colaborem entre si para compor as informações de um cliente, ou mesmo que operem nas informações destes, devem ser inseridas neste pacote ou depender deste. A separação dos dados de residência do Cliente

em uma entidade separada fornece uma visão mais correta e aproximada do mundo real e dos objetos.

#### 3.5.5 Pacote Agendamento

O pacote agendamento armazena as classes principais do sistema, a entidade Agendamento e Agenda, e por este motivo depende e é dependências de várias outras, existindo um forte acoplamento entre elas. Enquanto a entidade Agendamento representa um registro de serviço a ser realizado ou já realizado, a Agenda opera sobre ela de forma especial, mantendo os registros na ordem de execução correta.

#### 3.5.6 Pacote Relatórios

O pacote de relatórios, neste sistema, é um pacote especial por não conter entidades. Seu funcionamento consiste em obter informações de outros pacotes de forma adequada à solicitação do usuário.



### 3.6 PROGRAMAÇÃO E ESTRUTURAS DE DADOS DA APLICAÇÃO

Enquanto que os diagramas de classes representam um modelo orientado a objetos, o diagrama de casos de uso dirige o desenvolvimento do software por funcionalidades e os Diagramas de Entidade Relacionamento dizem como os dados são armazenados, é na fase de programação que as regras de negócio são aplicadas.

Este estudo não visa gerar um protótipo funcional, mas como é necessário demonstrar algumas restrições e casos especiais gerados pelas regras de negócio, alguns códigos de programação e conceitos abstratos serão apresentados a seguir.

#### 3.6.1 Estrutura de dados e Regras de Negócio

De acordo com a Encyclopædia Britannica (2014), em tradução livre, uma estrutura de dados é “uma maneira de armazenar dados para busca e recuperação de forma eficiente”.

A Ciência da Computação nos fornece os conceitos e definições de várias estruturas, sua implementação e variações. Entretanto, a ciência da computação fornece apenas a descrição técnica de cada uma, não pretendendo e nem oferecendo soluções universais para todos os casos. Por isso, é importante estudar quais as opções disponíveis, se elas atendem totalmente a necessidade ou se são necessários códigos adicionais.

##### 3.6.1.1 Fila

De acordo com a definição da CAELUM, “as Filas têm operações mais **restritas** do que as operações das Listas. Nas Filas, os elementos são adicionados na última posição e removidos da primeira posição. Nas Listas, os elementos são adicionados e removidos de qualquer posição.”. Sendo assim, temos que uma estrutura de fila tem a regra de FIFO (First-in First-out, que em inglês significa que o primeiro a entrar também é o primeiro a sair).

Em uma definição mais próxima do mundo real, a CAELUM novamente diz que “As filas são importantes pois elas determinam a ordem de atendimento das pessoas. As pessoas são atendidas conforme a posição delas na fila. O próximo a ser atendido é o primeiro da fila. Quando o primeiro da fila é chamado para ser atendido a fila “anda”, ou seja, o segundo passa a ser o primeiro, o terceiro passa a ser o segundo e assim por diante até a última pessoa.”

De acordo com nossa regra de negócio, devemos recuperar os registros de datas mais próximas primeiro e verificar se o elemento tem o mesmo horário, e portanto uma Fila simples não atende nossos requisitos.

#### 3.6.1.2 Fila Priorizada

Uma fila priorizada é como uma fila convencional, mas com uma regra de adicional: os elementos são inseridos na lista com uma marcação de prioridade, e elementos de menor prioridade devem ser “servidos” depois dos elementos de maior prioridade. Para alcançar o resultado esperado, os elementos podem ser ordenados durante a inserção, resultando assim em inserção de performance  $O(n)$ , remoção de  $O(1)$  e busca de performance  $O(n)$ .

No nosso estudo, pretendemos inserir Agendamentos em uma Agenda e recuperá-lo de acordo com a sua data de agendamento. Portanto, já temos o marcador de prioridade. Entretanto, nenhuma estrutura de dados convencional atenderá a todas nossas regras, e portanto será utilizado o conceito básico de fila priorizada mas adicionando nossas próprias regras.

#### 3.6.2 Classe Agenda como uma estrutura complexa

Para exemplificar como a estrutura pode ser implementada, será utilizada a linguagem de programação C#. O código principal, demonstrado em uma única imagem, e uma implementação do “esqueleto” das entidades foi fornecida e preenchida com valores estáticos.

Figura 3 – Código da Classe Agenda

```

public class Agenda {
    public Agenda() {}

    public int NumeroDeAgendamentos {
        get {
            return listaAgendamentos.Count;
        }
    }

    private List<Agendamento> listaAgendamentos = new List<Agendamento>();

    public void adicionaAgendamento(Agendamento agendamentoServico) {
        int insertionPos = 0;
        // encontra a melhor posição de inserção, onde elementoInserido.horario < prox.horario
        for(insertionPos = 0; insertionPos < listaAgendamentos.Count; insertionPos++) {
            Agendamento currentPosAgendamento = listaAgendamentos.ElementAt(insertionPos);

            // não podemos ter agendamento para o mesmo horário
            // TODO: Devemos checar por uma faixa de horário, considerando o tempo de serviço.
            if (agendamentoServico.HoraAgendamento == currentPosAgendamento.HoraAgendamento) {
                throw new HorarioAgendamentoException("Já existe um Agendamento para o mesmo horário.");
            }

            // se encontramos uma posição que o próximo agendamento é
            // "mais tarde" que o atual, é aqui que devemos inserir.
            if (agendamentoServico.HoraAgendamento < currentPosAgendamento.HoraAgendamento) {
                break;
            }
        }
        listaAgendamentos.Insert(insertionPos, agendamentoServico);
    }

    public Agendamento getPrimeiro() {
        return listaAgendamentos.FirstOrDefault();
    }

    public Agendamento get(int index) {
        return listaAgendamentos.ElementAt(index);
    }
}

```

Fonte: Produzido pelo pesquisador.

O código completo, incluindo o teste efetuado pelo programa, também é fornecido no Quadro 1, código este que, quando compilado, produz a saída da Figura 4.

Quadro 1 – Código fonte do programa

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.Text;
using System.Threading.Tasks;

namespace PROJETO_PETSHOP
{
    class Program
    {
        static void Main(string[] args)
        {
            Servico servico = new Servico();
            servico.Nome = "Tosa";
            servico.Descricao = "Consiste em um tratamento com direito a uma tosa higiênica e hidratação.";
            servico.Valor = 40.0;

            TipoAnimal tipo = new TipoAnimal();
            tipo.Descricao = "Cachorro";

            Raca raca = new Raca();

```

```

raca.Descricao = "Poodle";
raca.TipoAnimal = tipo;

Animal pet = new Animal();
pet.Nome = "Maycon";
pet.Idade = 4;
pet.Raca = raca;
pet.Sexo = Sexo.MACHO;

Agendamento agendamentoServico = new Agendamento();
agendamentoServico.HoraAgendamento = DateTime.Now;
agendamentoServico.Servico = servico;
agendamentoServico.Animal = pet;

Agenda agenda = new Agenda();

try {
    Console.WriteLine("Tentando agendar...");
    agenda.adicionaAgendamento(agendamentoServico);
    Console.WriteLine("Agendamento adicionado.");
    Console.WriteLine("Tentando agendar novamente o mesmo lançamento...");
    agenda.adicionaAgendamento(agendamentoServico);
    Console.WriteLine("Agendamento adicionado novamente.");
} catch (HorarioAgendamentoException e) {
    Console.WriteLine("Não foi possível agendar: " + e.Message);
}

Agendamento primeiro = agenda.getPrimeiro();

Console.Write(
    string.Format(
        "O primeiro:\n"+
        "\tEstá marcado para {0}\n"+
        "\tdo pet {1} de raça {2}\n"+
        "\tpara execução do serviço de {3}\n"+
        "\tno valor de R$ {4:#.00}.\n",
        primeiro.HoraAgendamento.ToString(),
        primeiro.Animal.Nome,
        primeiro.Animal.Raca.Descricao,
        primeiro.Servico.Nome,
        primeiro.Servico.Valor
    )
);
}

public enum Sexo {
    MACHO, FEMEA
}

public class Agenda {
    public Agenda() {}

    public int NumeroDeAgendamentos {
        get {
            return listaAgendamentos.Count;
        }
    }

    private List<Agendamento> listaAgendamentos = new List<Agendamento>();

    public void adicionaAgendamento(Agendamento agendamentoServico) {
        int insertionPos = 0;
        // encontra a melhor posição de inserção, onde elementoInserido.horario < prox.horario
        for (insertionPos = 0; insertionPos < listaAgendamentos.Count; insertionPos++) {
            Agendamento currentPosAgendamento = listaAgendamentos.ElementAt(insertionPos);

            // não podemos ter agendamento para o mesmo horário
            // TODO: Devemos checar por uma faixa de horário, considerando o tempo de serviço.
            if (agendamentoServico.HoraAgendamento == currentPosAgendamento.HoraAgendamento) {
                throw new HorarioAgendamentoException("Já existe um Agendamento para o mesmo
horário.");
            }
        }
    }
}

```

```

        // se encontramos uma posição que o próximo agendamento é
        // "mais tarde" que o atual, é aqui que devemos inserir.
        if (agendamentoServico.HoraAgendamento < currentPosAgendamento.HoraAgendamento) {
            break;
        }
    }
    listaAgendamentos.Insert(insertionPos, agendamentoServico);
}

public Agendamento getPrimeiro() {
    return listaAgendamentos.FirstOrDefault();
}

public Agendamento get(int index) {
    return listaAgendamentos.ElementAt(index);
}
}

public class Agendamento {
    public Animal Animal { get; internal set; }
    public DateTime HoraAgendamento { get; internal set; }
    public Servico Servico { get; internal set; }
}

public class Animal {
    public int Idade { get; internal set; }
    public string Nome { get; internal set; }
    public Raca Raca { get; internal set; }
    public object Sexo { get; internal set; }
}

public class Raca {
    public string Descricao { get; internal set; }
    public TipoAnimal TipoAnimal { get; internal set; }
}

public class Servico {
    public string Descricao { get; internal set; }
    public string Nome { get; internal set; }
    public double Valor { get; internal set; }
}

public class TipoAnimal {
    public string Descricao { get; internal set; }

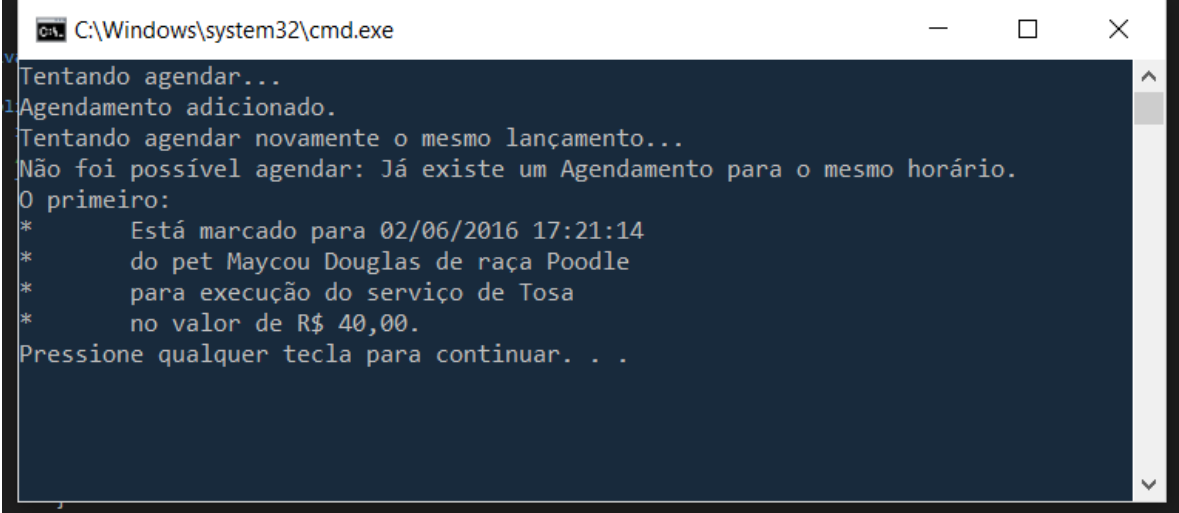
    internal void adicionaRaca(Raca raca) {
        throw new NotImplementedException();
    }
}

public class Usuario {
    public string Login { get; internal set; }
    public string Nome { get; internal set; }
    public string Senha { get; internal set; }
}
}

[Serializable]
internal class HorarioAgendamentoException : Exception {
    public HorarioAgendamentoException(string message) : base(message) {}
}
}

```

Figura 4 – Saída da execução do programa

A screenshot of a Windows command prompt window. The title bar at the top reads "C:\Windows\system32\cmd.exe" and includes standard minimize, maximize, and close buttons. The command prompt itself has a dark blue background with white text. The output of the program is as follows:

```
Tentando agendar...
1 Agendamento adicionado.
Tentando agendar novamente o mesmo lançamento...
Não foi possível agendar: Já existe um Agendamento para o mesmo horário.
O primeiro:
*   Está marcado para 02/06/2016 17:21:14
*   do pet Maycou Douglas de raça Poodle
*   para execução do serviço de Tosa
*   no valor de R$ 40,00.
Pressione qualquer tecla para continuar. . .
```

Fonte: Produzido pelo pesquisador.

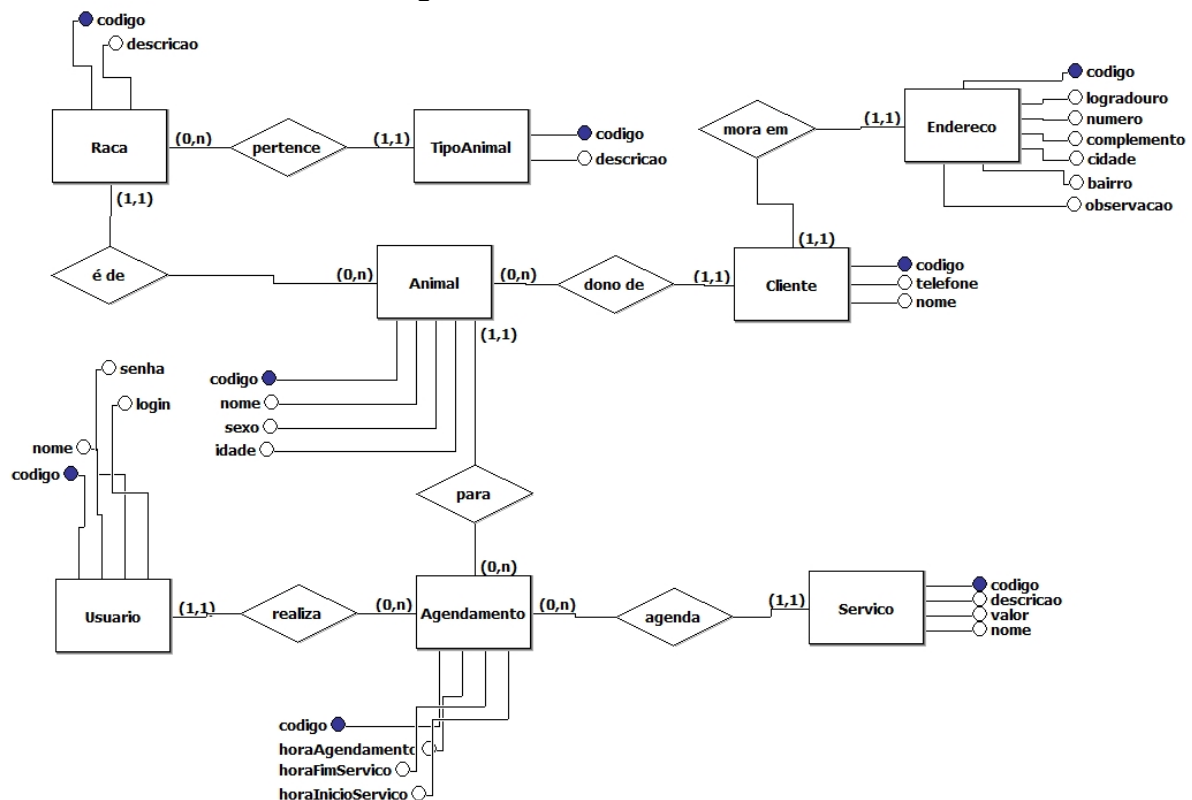
### 3.7 MODELO DE BANCO DE DADOS

Modelo de banco de dados é uma descrição, em forma textual, gráfica e programática, dos tipos de informações que estão armazenadas em um banco de dados. Este modelo não se preocupa com quais dados estão armazenados, apenas especifica quais informações devem ser armazenadas e como devem ser organizadas. É possível descrever os modelos em diferentes níveis de abstração e com diferentes objetivos.

#### 3.7.1 Modelo Conceitual

Um modelo conceitual representa dados que devem ser armazenados no banco, se aproximando e se preocupando mais com o negócio da empresa e menos com a implementação do sistema. Para apresentar o sistema deste estudo, uma representação gráfica foi gerada, que pode ser vista na figura a seguir.

Figura 5 – Modelo Conceitual

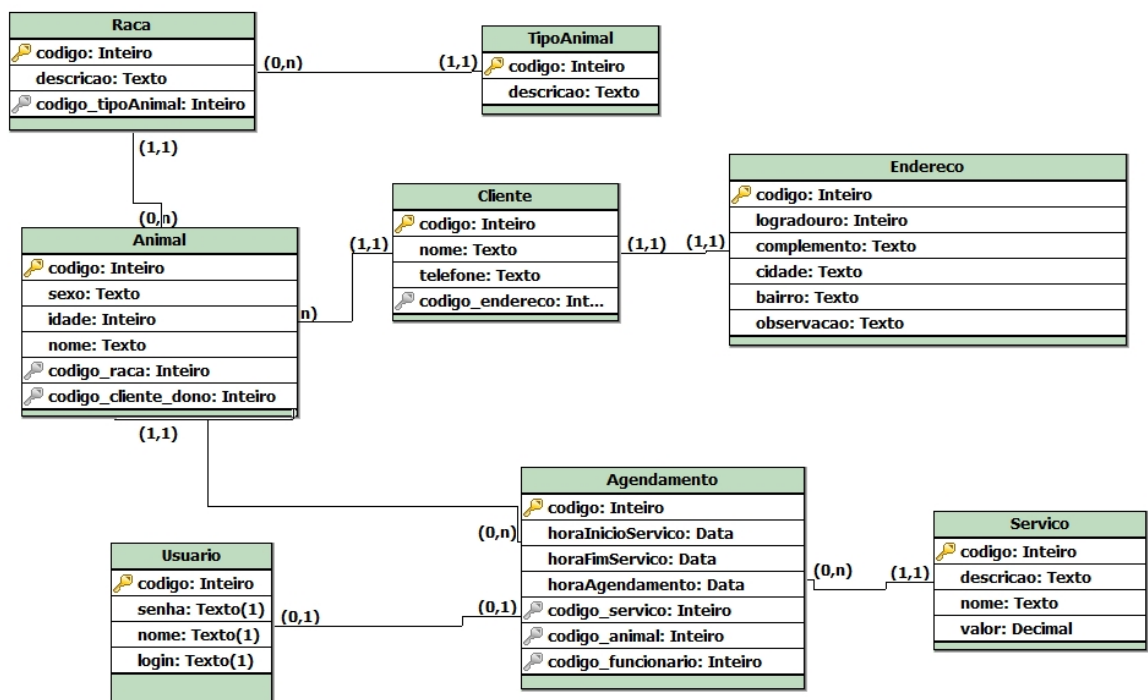


Fonte: Produzido pelo pesquisador.

### 3.7.2 Modelo Lógico

Um modelo de banco de dados lógico descreve as estruturas que serão utilizadas no banco de dados para armazenamento das informações, explicitando seus relacionamentos e integridade relacional, mas não opina ainda sobre os detalhes específicos de implementação nem tecnologia a ser utilizada. Uma representação gráfica, feita como um refinamento do Modelo Conceitual, foi produzida e pode ser vista na figura a seguir.

Figura 6 – Modelo Lógico



Fonte: Produzido pelo pesquisador.



### 3.8 MICROCOMPUTADORES E PROCESSOS DE NEGÓCIO

A revolução digital, que mudou o mundo todo desde o a década de 90, alterou significativamente as relações econômicas e empresariais. Com a importância cada vez maior de se fornecer à informação e facilidade de comunicação, as redes de computadores estão projetadas para crescer indefinidamente. É inegável que, assim como o maquinário trouxe vantagem para a indústria na revolução industrial, a informatização trouxe recursos que hoje são indispensáveis para se fornecer serviços a custos administráveis.

A adoção de uma plataforma baseada em microcomputadores se estiver em conformidade com as regras da empresa, garante benefícios na velocidade, segurança e precisão dos processos executados. Entretanto, é necessário levantar alguns pontos e discutir sobre eles para que esteja claro as vantagens e desvantagens.

#### 3.8.1 Vantagens

##### 3.8.1.1 Concorrência

Em sistemas computacionais, concorrência é quando existe acesso simultâneo de duas pessoas, sistemas ou processos a um mesmo recurso.

A partir da visão externa, de alto nível, podemos destacar a importância desta característica dos sistemas informatizados que, se bem programados, permitem que dois funcionários trabalhem em cima dos mesmos dados ou repositório de dados e não precisem se preocupar com a integridade ou sincronização das informações.

##### 3.8.1.2 Proximidade do cliente

Em um sistema seguro, é possível que o cliente tenha interação direta com os dados da empresa, podendo consultar, inserir, alterar e remover dados gerados por ele próprio ou disponibilizados para clientes, de acordo com as permissões definidas pelas regras de negócio. Tal exposição pode ser feita através

de sistemas web ou terminais locais, este primeiro mais popular atualmente pela rapidez das internet de banda larga.

### 3.8.2 Desvantagens

#### 3.8.2.1 Segurança

Problemas com segurança englobam várias deficiências na implementação de um sistema computacional, onde a falha pode ser proveniente do desenvolvimento incorreto de um software encomendado, de um terceiro ou mesmo do sistema operacional.

#### 3.8.2.2 Custo inicialmente alto

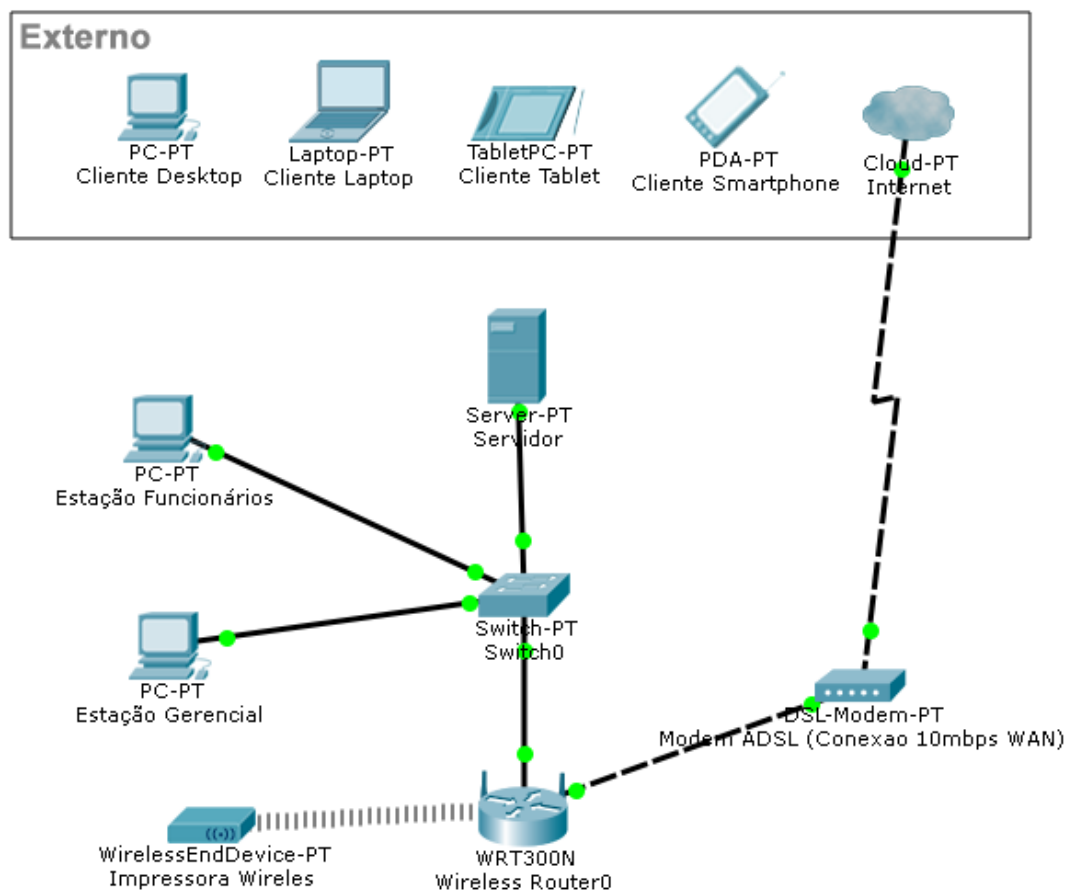
Em relação a uma abordagem tradicional, o custo de aquisição de hardware e desenvolvimento de software e, em especial o custo de software, pode ser alto e proibitivo para empresas de pequeno porte.

### 3.9 AMBIENTE E HARDWARE

Assim como os cuidados necessários no desenvolvimento do sistema, é preciso dimensionar os requisitos de software e especificar em quais condições devem ser feitas sua implementação para que ele se comporte adequadamente, questões estas que não podem ser respondidas apenas ao fim do desenvolvimento sob risco de prejuízo financeiro com o desenvolvimento do software, compra de hardware ou até dos dois.

Para que o software de PetShop cumpra os requisitos e tire proveito do esforço de desenvolvimento, é necessário que alguns componentes mínimos sejam fornecidos e interligados. Como guia e instrução de implementação, este estudo produziu mais um artefato gráfico que ilustra a organização da rede e seus equipamentos, representado na figura a seguir.

Figura 7 – Simulação de rede



Fonte: Produzido pelo pesquisador.

## 4 CONCLUSÃO

Este estudo foi feito com o objetivo de fornecer inteligência e dados sobre as regras de negócio e requisitos de desenvolvimento de um sistema de PetShop, como também demais aspectos da análise e planejamento de um sistema concreto, dimensionável e delimitado.

Durante todo o estudo foi possível manter as informações dentro do que foi proposto inicialmente, uma quantidade considerável de artefatos e gráficos foi produzida, e por este motivo as informações produzidas são valiosas para o enriquecimento do conhecimento de engenheiros de software, analistas de sistemas, demais profissionais de TI e até empreendedores do ramo de PetShops.

Este estudo, sendo multidisciplinar e abordando vários aspectos de um sistema, pôde ser coerente ao produzir informações que convergiram para o mesmo fim.

O processo de obtenção de informações foi de grande ajuda para a aprimorar o conhecimento dos envolvidos como pesquisadores e por à prova suas capacidades profissionais já existentes.

## REFERÊNCIAS

BRITANNICA, **The Editors of Encyclopædia. Data structure, Computer Science.** Publicação em inglês sobre estruturas de dados. Disponível em: <<http://global.britannica.com/technology/data-structure>> Acesso em: 17 mai. 2016.

BOOCH, Grady. **UML: guia do usuário.** Rio de Janeiro - RJ: Campus, 2000.

LIMA, Adilson da Silva. **UML 2.0: Do requisito a solução.** Editora Érica, 2005.

PRESSMAN, Roger S. **Engenharia de Software.** São Paulo - SP: Makron, 1995.

SILVA, Ricardo Pereira. **UML2 em Modelagem Orientada a Objetos.** Florianópolis: Visual Books, 2007.

MACORATTI, José Carlos. **UML – Conceitos Básicos II.** Disponível em:<[http://www.macoratti.net/vb\\_uml2.htm](http://www.macoratti.net/vb_uml2.htm) > Acesso em: 17 mai. 2016.

LARMAN, Craig. **Utilizando UML e Padrões.** Brasil. Bookman (edição digital), 2011