ENUCOMP

ENCONTRO UNIFICADO DE COMPUTAÇÃO

08 A 11 DE NOVEMBRO

ANAIS ELETRÔNICOS ENUCOMP 2016

ORGANIZAÇÃO:

Eyder Franco Sousa Rios

Rodrigo Augusto Rocha Souza Baluz

1ªEdição
Editora:
FUESPI

INFORMAÇÕES:
WWW.ENUCOMP.COM.BR

Promoção

Realização

Patrocinadores

Apoio

TERESINA- PI
2016

ENUCOMP

ENCONTRO UNIFICADO DE COMPUTAÇÃO

ANAIS ELETRÔNICOS ENUCOMP 2016

1ª Edição

EDITORA:

Fundação Universidade Estadual do Piauí-FUESPI

ORGANIZAÇÃO:

Prof. Dr. Eyder Franco Sousa Rios

Prof. MSc. Rodrigo Augusto Rocha Souza Baluz

PROMOÇÃO:

NUPEC DELTA

REALIZAÇÃO:

TERESINA – PI
2016

Ficha Catalográfica elaborada pela Bibliotecária
Christiane Maria Montenegro Sá Lins CRB/3 - 952

E56a

ENCONTRO UNIFICADO DE COMPUTAÇÃO. (7.: 2016: Teresina, PI).

Anais do IX ENUCOMP 2016, Teresina, PI, 08 a 11 de novembro de 2016: [recurso eletrônico]/ Organização [de] Eyder Franco Sousa Rios e Rodrigo Augusto R. S. Baluz. – Parnaíba: FUESPI, 2016.

250 p.: I1.

ISSN.: 978-85-8320-173-1

1. Computação gráfica. 2. Redes de computadores. I. Rios, Eyder Franco Sousa (org.) II. Baluz, Rodrigo Augusto R. S. (org.) III. Título.

CDD 001.642

PREFÁCIO

O Encontro Unificado de Computação (ENUCOMP) nasceu da integração de cursos na área da Computação e Informática na cidade de Parnaíba, litoral do estado do Piauí. Suas propostas foram pautadas na contribuição para a troca de experiências, buscando a união dos acadêmicos; no fortalecimento da parceria no desenvolvimento da educação e da informática; e no incentivo à produção de trabalhos científicos ligados à área de tecnologia.

O ENUCOMP vem se firmando como um simpósio regional de grande porte na área tecnológica e tem como objetivo promover a discussão em torno de temas relevantes e atuais da Computação. Em 2013, passou a trabalhar o tema da integração entre indústria produtiva, o mercado profissional e seus anseios, inovação e empreendedorismo, e academia – centro intelectual – motivando assim o desenvolvimento de pesquisas aplicadas.

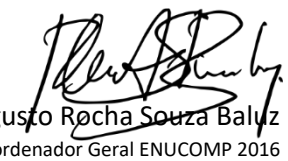
A edição 2015 trouxe a comunidade à Inovação e o Empreendedorismo como elementos norteadores do evento. Parceria mantida com o Serviço de Apoio às Micros e Pequenas Empresas do Piauí – SEBRAE permitiu, em paralelo a programação do ENUCOMP 2015, a Feira do Empreendedorismo 2015, um dos maiores e mais esperados eventos de empreendedorismo do Brasil. Dentre seus produtos foram realizados a Corrida de Startups, direcionado a novos produtos para o segmento do Turismo no Piauí, e a Zona Empreendedora, com espaço para validação de modelos de negócios e apresentações de *pitchs*.

Promovido anualmente pelo Núcleo de Pesquisa e Extensão em Computação do Delta do Parnaíba (NUPEC DELTA), grupo de pesquisa e extensão da Universidade Estadual do Piauí (UESPI), campus de Parnaíba-PI, o ENUCOMP 2016 foi realizado como evento satélite do 22º Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia), evento promovido anualmente pela Sociedade Brasileira de Computação (SBC). Realizado de 08 a 11 de novembro de 2016, na cidade de Teresina-PI, uma cidade aconchegante, quente e acolhedora no nordeste do Brasil. Sua programação contou com sessões técnicas de apresentação de artigos científicos, palestras nacionais e internacionais e minicursos. Os temas de interesse para submissão de trabalhos foram divididos em 4 (quatro) trilhas: 1. Engenharia de Software, 2. Redes de Computadores, 3. Visão Computacional e Processamento de Imagens, 4. Automação e Sistemas Inteligentes.

Em sua nona edição, foram recebidos 29 artigos científicos de diversas instituições brasileiras dos estados do Piauí, Ceará, São Paulo, Bahia e Paraíba. Nosso Comitê de Programa foi formado por 36 pesquisadores de São Paulo, Espírito Santo, Paraíba, Minas Gerais, Rio de Janeiro, Ceará e Piauí. Os 3 (três) melhores artigos avaliados pelo comitê garantiram sua publicação em uma edição especial da Revista Brasileira de Computação Aplicada (indexada CAPES Qualis B5).

Por último, gostaríamos de agradecer ao Corpo Editorial da RBCA (Revista Brasileira de Computação Aplicada) pela parceria na publicação dos melhores artigos do nosso evento. A Comissão Especial de Sistemas Multimídia e Web que cuida da organização do WebMedia, pelo convite e parceria na realização conjugada dos eventos. O nosso muito obrigado também aos palestrantes, aos ministrantes de minicursos e aos membros da equipe de apoio e do comitê de programa, por acreditarem em nosso evento. O trabalho voluntário realizado por vocês foi de fundamental importância para o sucesso do ENUCOMP. Desejamos que o evento possa trazer frutos para o trabalho de todos.

Até nossa edição de 10 anos!



Rodrigo Augusto Rocha Souza Baluz
Coordenador Geral ENUCOMP 2016

COMISSÃO ORGANIZADORA

ORGANIZAÇÃO GERAL

Prof. Rodrigo Augusto Rocha Souza Baluz (UESPI/Parnaíba)
Prof. Ricardo de Andrade Lira Rabelo (UFPI/Teresina)
Prof. Fábio de Jesus Lima Gomes (IFPI/Teresina)
Prof. Eyder Franco Sousa Rios (UESPI/Parnaíba) Chair do Comitê

COMISSÃO DE ORGANIZAÇÃO

Prof. Carlos Giovanni Nunes de Carvalho (UESPI/Teresina)
Prof. Harilton da Silva Araújo (Faculdade Estácio de Sá-CEUT/Teresina)
Prof. Athanio de Souza Silveira (IFPI/Parnaíba)
Prof. Antonio dos Santos Sousa (IFPI/Parnaíba)
Prof. Ely Bezerra Silva Júnior (FMN/Parnaíba)
Prof. Clodoaldo Brasilino Leite Neto (IFPI/Parnaíba)
Prof. Thiago Carvalho de Sousa (UESPI/Teresina)
Prof. Sérgio Barros de Sousa (UESPI/Parnaíba)
Prof. Gildário Dias Lima (UFPI/Parnaíba)
Prof. Francisco das Chagas Rocha (UESPI/Parnaíba)
Prof. Átila Rabelo Lopes (UESPI/Parnaíba)
Prof. Leinylson Fontelene Pereira (FMN/Parnaíba)
Prof. Henrique Rocha Fontenele (FMN/Parnaíba)
Prof. Dário Brio Calçada (UESPI/Parnaíba)
Profa. Lianna Mara Castro Duarte (UESPI/Teresina)
Profa. Cornélia Janayna Pereira Passarinho (UFPI/Teresina)
Prof. José Flávio Gomes Barros (IFMA/Caxias)
Prof. José Wilker Pereira Luz (IFMA/Caxias)
Prof. Paulo César Coutinho (Polo Tecnológico de Parnaíba/SEBRAE)
Prof. Francisco Gerson Amorim de Meneses (IFPI/Parnaíba)

COMITÊ DE PROGRAMA

Prof. Dr. Gilvandenys Leite Sales (IFCE)
Prof. Dr. Paulo Benicio Melo de Sousa (FFB)
Prof. Dr. André Castelo Branco Soares (UFPI - PPGCC)
Prof. Dr. Ivan Saraiva Silva (UFPI - PPGCC)
Prof. Dr. Raimundo Santos Moura (UFPI - PPGCC)
Prof. Dr. Pedro de Alcântara dos Santos Neto (UFPI - PPGCC)
Prof. Dr. Kelson Rômulo Teixeira Aires (UFPI - PPGCC)
Prof. Dr. Ricardo de Andrade Lira Rabêlo (UFPI - PPGCC)
Prof. Dr. Vinícius Ponte Machado (UFPI - PPGCC)
Prof. Dr. Alberto Sampaio Lima (UFC - Campus Quixadá)
Prof. Dr. Flávio Rubens de Carvalho Sousa (UFC DETI)
Prof. Dr. Wladimir Araujo Tavares (UFC - Campus Quixadá)

SUMÁRIO

MINICURSOS

ARTIGOS

Capítulo 1

Construindo uma Nuvem Privada com Openstack

Julio Cesar Damasceno (julio.damasceno@ufrpe.br)
Josino Rodrigues Neto (josino.neto@palmares.ifpe.edu.br)
Vinicius Cardoso Garcia (vcg@cin.ufpe.br)
Rodrigo Elia Assad (rodrigo.assad@ufrpe.br)

Resumo

A computação em Nuvem ou, Cloud Computing, tem-se mostrado um paradigma bastante promissor, pois existe a possibilidade de se pagar apenas pelos recursos utilizados de maneira flexível onde os mesmos podem escalar para mais ou para menos. A vantagem de se optar por uma nuvem privada, onde os recursos são disponibilizados apenas para um grupo restrito de usuários, é motivada pelo fato que a sua construção pode ser feita a partir de hardware atual com um custo razoável. Este minicurso tem por objetivo apresentar o Openstack como solução para construir uma nuvem privada.

1.1. Introdução

OpenStack é um sistema operacional de nuvem open source que vem ganhando espaço na comunidade de software livre e na indústria. Criado pela NASA e Rackspace Hosting, o projeto tem recebido investimentos significantes de líderes da indústria como HP, Huawei, IBM, Intel e Red Hat.

O uso da plataforma Openstack vem crescendo no Brasil, atualmente a empresa UOL utiliza o módulo de Cloud Storage do OpenStack e no âmbito público, o SERPRO está montando sua nuvem coorporativa baseada em OpenStack. Uma pesquisa encomendada pela SUSE, provedora de infraestrutura de armazenamento, cloud e open source Linux, apontou que 81% dos profissionais dos setores de tecnologia da informação de grandes empresas têm a intenção de utilizar OpenStack.

No cenário acadêmico OpenStack tem se tornado cada vez mais presente em pesquisas que investigam computação em nuvem e infraestrutura como serviço. As universidades brasileiras estão se destacando, não apenas no uso do Openstack, mas também contribuindo no desenvolvimento do código fonte da plataforma. No OpenStack Summit 2016 de Austin, Texas-EUA, maior encontro da comunidade, a UFCG foi reconhecida como a universidade brasileira que mais colabora com OpenStack no mundo.

Este minicurso tem como base fornecer aos participantes os aspectos fundamentais do funcionamento do Open Stack, uma descrição de sua arquitetura e o funcionamento de seus componentes. Os alunos do minicurso serão apresentados aos tópicos emergenes relacionados a Cloud Computing e a um passo a passo para montar uma nuvem privada na forma de Infraestrutura como Serviço (IaaS).

1.2. Virtualização

Virtualização é uma técnica que permite a execução simultânea de dois ou mais ambientes (sistemas operacionais) distintos e isolados em uma mesma máquina. Esse conceito de virtualização relembra os antigos mainframes, que deviam ser compartilhados por vários usuários em ambientes de aplicação completamente diferentes. Essa realidade da década de 1970 foi em grande parte superada nos anos de 1980 e 1990, com o surgimento dos computadores pessoais. No entanto, atualmente há uma onda crescente de interesse sobre as técnicas de virtualização.

Nos dias de hoje, o interesse na virtualização não se atém somente ao fato de permitir o uso de um mesmo sistema por vários usuários concomitantemente, mas os principais interesses são as vantagens oferecidas por esse tipo de ambiente, são elas: segurança, confiabilidade e disponibilidade, custo, adaptabilidade, balanceamento de carga e suporte a aplicações legadas.

1.2.1. Definições e Conceitos

Os primeiros conceitos que devemos ter em relação à virtualização são os conceitos de instruções privilegiadas e não privilegiadas. Essas instruções fazem parte do conjunto de instruções de uma arquitetura. As instruções não-privilegiadas são aquelas que não modificam a alocação ou o estado de recursos compartilhados por vários processos simultâneos, tais como processadores, memória principal e registradores especiais. Em contrapartida, temos as instruções privilegiadas, que podem alterar o estado e a alocação desses recursos.

Um computador pode operar em dois modos distintos, o modo de usuário ou o de supervisor. O modo de usuário, também chamado de espaço de aplicação, é o modo no qual as aplicações normalmente são executadas. Neste modo, não é possível executar as instruções privilegiadas, que são restritas ao modo de supervisor.

O modo de supervisor tem o controle total sobre a CPU, podendo executar todas as instruções do conjunto de instruções do processador, tanto as não-privilegiadas como as privilegiadas. O sistema operacional é executado neste modo. Antes do sistema operacional passar o controle da CPU para uma aplicação do usuário, o bit de controle de modo é configurado para o modo de usuário.

Vale lembrar que na arquitetura x86, existem quatro níveis de privilégio, que são chamados de *rings*. Os *rings* são numerados de 0 a 3, nos quais o nível 0 é o que tem maior privilégio na execução de instruções, por isso, os sistemas operacionais são executados com esse nível de privilégio.

Já em um ambiente virtualizado, temos que definir mais dois conceitos, os de sistema operacional hospedeiro e o de sistema operacional visitante. O sistema operacional

hospedeiro (*Host Operating System*), refere-se ao SO nativo da máquina na qual ocorrerá a virtualização. Já o sistema operacional visitante (*Guest Operating System*), refere-se ao sistema operacional que é executado sobre a máquina virtual. Uma máquina na qual é feita a virtualização pode-se contar com apenas um SO hospedeiro sendo executado por vez. No entanto, podem ser executados diversos SOs visitantes simultaneamente.

O próximo conceito a ser discutido é de vital importância para o entendimento da virtualização. O conceito em questão é o do VMM (*Virtual Machine Monitor*), ou seja, Monitor de Máquina Virtual, também conhecido por *Hypervisor*. O *hypervisor* provê uma camada de abstração entre o hardware e o sistema operacional que está executando conforme é apresentado na Figura 1.1.

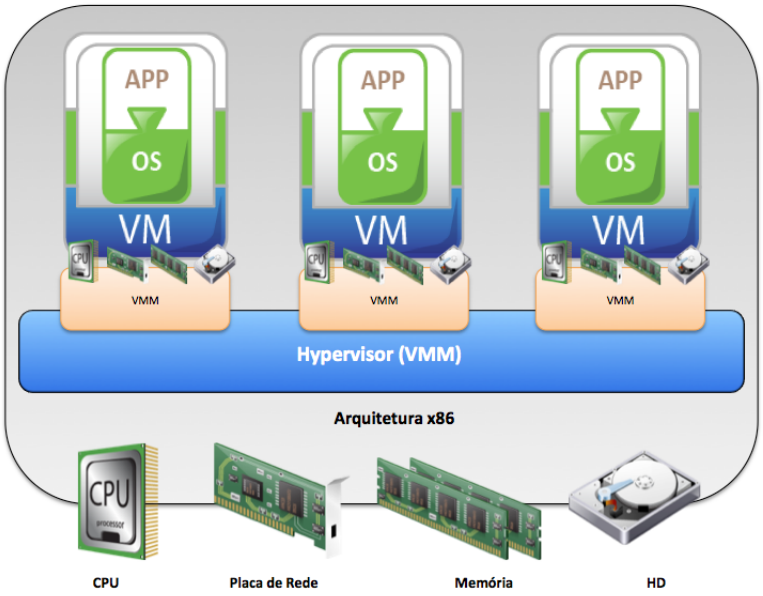


Figura 1.1: Relacionamento das Máquinas Virtuais e o VMM

O Virtual Machine Monitor é um componente de software que hospeda as máquinas virtuais [12]. O VMM é responsável pela virtualização e controle dos recursos compartilhados entre as máquinas virtuais, tais como, processadores, dispositivos de entrada e saída, memória, armazenamento, etc. Também é função do VMM escalonar qual máquina virtual vai executar a cada momento, semelhante ao escalonador de processos do Sistema Operacional [9].

O VMM é executado no modo de supervisor, no entanto as máquinas virtuais são executadas em modo de usuário. Como as máquinas virtuais são executadas em modo de usuário, quando estas tentam executar uma instrução privilegiada, é gerada uma interrupção e o VMM se encarrega de emular a execução desta instrução.

1.2.2. Vantagens e Desvantagens

Existem diversas vantagens na virtualização, segundo [12], a seguir serão citadas as principais:

- Segurança: Usando máquinas virtuais, pode ser definido qual é o melhor ambiente

para executar cada serviço, com diferentes requerimentos de segurança, ferramentas diferentes e o sistema operacional mais adequado para cada serviço. Além disso, cada máquina virtual é isolada das demais. Usando uma máquina virtual para cada serviço, a vulnerabilidade de um serviço não prejudica os demais;

- Confiança e disponibilidade: A falha de um software não prejudica os demais serviços;
- Custo: A redução de custos é possível de ser alcançada com a consolidação de pequenos servidores em outros mais poderosos. Essa redução pode variar de 29 a 64 [12];
- Adaptação às diferentes cargas de trabalho: Variações na carga de trabalho podem ser tratadas facilmente. Ferramentas autônomas podem realocar recursos de uma máquina virtual para a outra;
- Balanceamento de carga: Toda a máquina virtual está encapsulada no VMM. Sendo assim é fácil trocar a máquina virtual de plataforma, a fim de aumentar o seu desempenho;
- Suporte a aplicações legadas: Quando uma empresa decide migrar para um novo Sistema Operacional, é possível manter o sistema operacional antigo sendo executado em uma máquina virtual, o que reduz os custos com a migração. Vale ainda lembrar que a virtualização pode ser útil para aplicações que são executadas em hardware legado, que está sujeito a falhas e tem altos custos de manutenção. Com a virtualização desse hardware, é possível executar essas aplicações em hardware mais novos, com custo de manutenção mais baixo e maior confiabilidade.

Por outro lado, existem as desvantagens da virtualização, sendo as principais:

- Segurança: Segundo Neil MacDonald, especialista de segurança da Gartner, as máquinas virtuais são menos seguras que as máquinas físicas devido ao VMM [6]. Este ponto é interessante, pois se o sistema operacional hospedeiro tiver alguma vulnerabilidade, todas as máquinas virtuais que estão hospedadas nessa máquina física estão vulneráveis, já que o VMM é uma camada de software, portanto, como qualquer software, está sujeito a vulnerabilidades;
- Gerenciamento: Os ambientes virtuais necessitam ser instanciados, monitorados, configurados e salvos [3]. Existem produtos que fornecem essas soluções, mas esse é o campo no qual estão os maiores investimentos na área de virtualização, justamente por se tratar de um dos maiores contra-tempos na implementação da virtualização. Vale lembrar que o VMware é a plataforma mais flexível e fácil de usar, mas ainda apresenta falhas que comprometem a segurança, assim como as demais plataformas [3];
- Desempenho: Atualmente, não existem métodos consolidados para medir o desempenho de ambientes virtualizados. No entanto, a introdução de uma camada extra de software entre o sistema operacional e o hardware, o VMM ou *hypervisor*, gera

um custo de processamento superior ao que se teria sem a virtualização. Outro ponto importante de ressaltar é que não se sabe exatamente quantas máquinas virtuais podem ser executadas por processador, sem que haja o prejuízo da qualidade de serviço.

1.2.3. Técnicas de Virtualização

Existem três técnicas para lidar com instruções privilegiadas virtualizando CPU na arquitetura x86. A Figura 1.2 mostra uma comparação entre as técnicas.

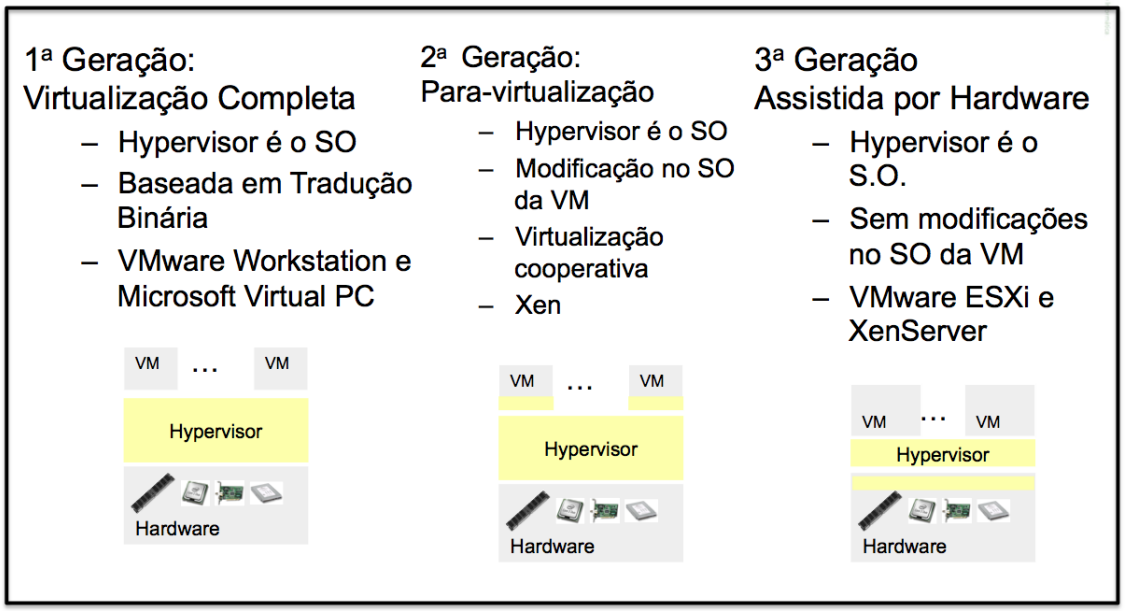


Figura 1.2: Comparação das Técnicas de Virtualização

A virtualização completa tem por objetivo fornecer ao sistema operacional visitante uma réplica do hardware subjacente. Dessa forma, o sistema operacional visitante é executado sem modificações sobre o monitor de máquina virtual (VMM), o que traz alguns inconvenientes. O primeiro é que o número de dispositivos a serem suportados pelo VMM é extremamente elevado.

Para resolver esse contratempo, as implementações da virtualização total usam dispositivos genéricos, que funcionam bem para a maioria dos dispositivos disponíveis, mas não garantem o uso da totalidade de sua capacidade. Outro inconveniente da virtualização total é o fato do sistema operacional visitante não ter conhecimento de que está sendo executado sobre o VMM, então as instruções executadas pelo sistema operacional visitante devem ser testadas pelo VMM para que depois sejam executadas diretamente no hardware, ou executadas pelo VMM e simulada a execução para o sistema visitante. Por fim, o último inconveniente da virtualização total é o fato de ter que contornar alguns problemas gerados pela implementação dos sistemas operacionais, já que esses foram implementados para serem executados como instância única nas máquinas física, não disputando recursos com outros sistemas operacionais. Um exemplo desse último inconveniente é o uso de paginação na memória virtual, pois há disputa de recursos entre diversas instâncias de sistemas operacionais, o que acarreta em uma queda do desempenho [3].

A para-virtualização é uma alternativa à virtualização total. Nesse modelo de virtualização, o sistema operacional é modificado para chamar o VMM sempre que executar uma instrução que possa alterar o estado do sistema, uma instrução sensível. Isso acaba com a necessidade de o VMM testar instrução por instrução, o que representa um ganho significativo de desempenho. Outro ponto positivo da para-virtualização é que os dispositivos de hardware são acessados por *drivers* da própria máquina virtual, não necessitando mais do uso de *drivers* genéricos que inibiam o uso da capacidade total do dispositivo.

Embora a para-virtualização apresentasse um ganho de desempenho significativo frente à virtualização total, essa disparidade tem sido superada devido à presença de instruções de virtualização nos processadores Intel e AMD, que favorecem a virtualização total. Embora tenham sido desenvolvidas para o mesmo propósito, foram criadas de maneira independente. Por esse motivo, há alguns problemas na portabilidade de máquinas virtuais de uma arquitetura Intel para a arquitetura AMD e vice-versa.

A virtualização assistida por hardware é conseguida através da utilização de CPU "hypervisor-aware" para lidar com instruções privilegiadas, onde o suporte à virtualização é implementado dentro do processador. A principal vantagem é a redução da sobrecarga de virtualização no *hypervisor* na virtualização completa e na para-virtualização. É implementada pela tecnologias AMD-V e Intel VT nas arquiteturas de processadores x86.

A virtualização tem estado presente nos *datacenters* por vários anos como uma tecnologia bem sucedida para consolidação de servidores. A virtualização fornece a base para a construção de um ambiente de nuvem, pois a aumenta a agilidade e flexibilidade.

1.3. Computação em Nuvem

Uma definição bem aceita de computação em nuvem, ou do inglês *cloud computing*, estabelecida pelo NIST (*National Institute of Standards and Technology*) assume que "Computação em nuvem é um modelo que permite o acesso ubíquo, conveniente e sob demanda de recursos computacionais compartilhados e configuráveis, tais como, redes, servidores, armazenamento, aplicações e serviços, que podem ser rapidamente provisionados e entregues ao usuário com um esforço mínimo de gerenciamento ou de interação com o provedor de serviços" [8].

Computação em nuvem é um modo de usar recursos computacionais apenas quando necessário pelo tempo que for necessário. Como a Internet é a base para a computação em nuvem, pois todos os serviços e recursos devem estar disponíveis a qualquer momento e em qualquer dispositivo, porém a decisão de colocar dados sensíveis na Internet pode criar um novo problema relacionado à privacidade e à segurança dos dados. Uma possibilidade para amenizar estes problemas é montar uma infraestrutura privada, onde apenas um grupo restrito de usuários terá acesso aos recursos desta nuvem.

1.3.1. Propriedades

Ainda segundo o NIST, Computação em Nuvem é composta por algumas características essenciais [8], dentre as quais destacam-se:

- Auto-atendimento sob demanda (*on-demand self-service*): o usuário pode acessar os recursos da cloud e requisitar, de acordo com a sua demanda, a quantidade de

recursos que será disponibilizada. O custo financeiro desta requisição do usuário poderá ser diretamente proporcional à quantidade de recursos requisitados. Se o usuário não puder dizer o que precisa, o ambiente perderá força para ser considerado como uma cloud;

- Amplo acesso à rede (*broad network access*): por ser utilizada por uma grande quantidade de usuários e por depender de infraestrutura de comunicação adequada para sua execução, o ambiente deve possuir capacidade suficientemente grande para receber acessos simultâneos provenientes de diferentes origens. Um investimento reduzido na infraestrutura de rede da cloud pode se tornar um gargalo para a sua utilização;

- Grupo de recursos (*resource pooling*): em uma nuvem, diversos recursos computacionais (como servidores) são disponibilizados para os usuários de forma transparente. O usuário requisita da nuvem o que ele deseja e a mesma retorna para ele o recurso requisitado. O usuário não precisa saber que servidores estão processando a sua tarefa, e muito menos onde esses servidores estão fisicamente (princípio de independência de localização);

- Elasticidade rápida (*rapid elasticity*): uma outra propriedade importante de uma cloud é a sua elasticidade; em suma, pode-se definir a elasticidade de uma cloud como sua capacidade de disponibilizar mais recursos à medida que isto for sendo requisitado pelo usuário (ou pela sua aplicação). O ambiente de cloud deve ser capaz de prover ao usuário, de forma rápida e padronizada, mecanismos para a requisição e disponibilização de mais recursos computacionais. Esta é uma das principais propriedades de uma Cloud; se a Cloud não tiver esta capacidade, sua própria caracterização como cloud fica comprometida;

- Serviços medidos (*measured services*): uma outra característica vital de uma nuvem é a mensuração de seus serviços. O modelo de negócio deste tipo de ambiente é baseado no modelo pague pelo uso (*pay-per-usage*); desta forma, devem ser disponibilizadas formas de se medir a utilização de recursos e serviços. Um dos temas atuais nessa propriedade da nuvem é a bilhetagem, e nele diversos esforços de pesquisa vêm sendo propostos. Se o ambiente de nuvem não tiver como medir e cobrar pela sua utilização, o modelo de negócio associado a este novo paradigma pode estar fadado ao insucesso, tendo em vista que as empresas não terão algoritmos efetivos para efetuar a cobrança dos seus clientes.

Um ambiente de nuvem deve possuir algumas características desejáveis. Uma primeira característica desejável em uma nuvem é a homogeneidade; se um conjunto homogêneo (produtos iguais ou similares) de recursos (softwares e hardwares) for utilizado , tarefas como instalação e gerenciamento serão largamente simplificadas. Por produtos similares, deve-se entender um conjunto de softwares compatíveis e configurações/marcas de recursos computacionais (como servidores) similares (preferencialmente iguais).

Uma segunda característica fundamental é o uso de virtualização para o melhor aproveitamento dos recursos computacionais. A utilização de virtualização no ambiente

de nuvem é uma tendência, pois obtem-se melhores resultados em comparação a outros ambientes, como *Grid Computing* [4].

A utilização de software de baixo custo, especialmente software livre, é outra característica marcante. O custo de implementação de uma ambiente de nuvem pode sofrer uma redução significativa pela utilização de tal artifício, e este baixo custo acaba refletindo em um menor preço pelo serviço disponibilizado ao usuário final.

Por fim, uma característica também desejável (e até necessária) em um ambiente de nuvem é a orientação a serviço. O provedor nuvem deve fornecer alguma forma de cobrar a utilização de seu ambiente para os clientes; disponibilizando seus recursos como serviço. Regras podem ser colocadas, como SLA (*Service Level Agreement*), para padronizar e legalizar o negócio entre provedores de serviço e clientes, trazendo mais confiabilidade para o modelo de negócio.

1.3.2. Modelos de Serviço

Considerando o contexto variado de aplicações para a nuvem, uma classificação atualmente bastante utilizada e difundida é baseada nos modelos de serviços ofertados ao usuário, conforme mostrado na Figura 1.3.

Figura 1.3: Principais modelos de serviços e seus usuários em computação em Nuvem

- IaaS (*Infrastructure-as-a-Service*): é o modelo que se encontra mais próximo do hardware/recursos físicos. Em geral, serviços neste modelo proveem uma API (*Application Programming Interface*) de gerenciamento para a utilização de um conjunto de recursos físicos de forma que o usuário tenha acesso a funcionalidades como configurações automatizadas do sistema operacional, escalabilidade sob demanda e armazenamento. Como exemplo deste tipo de cloud, pode-se citar o Eucalyptus, Amazon AWS e o OpenNebula [11, 2, 10];

- PaaS (*Platform-as-a-Service*): permite aos utilizadores adaptar aplicações legadas para o ambiente de nuvem ou desenvolver aplicativos nativos para nuvem (*cloud-aware*) utilizando linguagens de programação, serviços, bibliotecas e outras ferra-

mentas de desenvolvimento disponibilizadas pelo provedor. Um exemplo interessante é o *Google App Engine*[7], que oferece, não apenas recursos de armazenamento, mas também um ambiente para programação onde pode-se desenvolver e executar aplicações;

- SaaS (*Software-as-a-Service*): neste modelo os usuários podem executar aplicativos através de múltiplos dispositivos na infraestrutura em nuvem. Exemplos de serviços disponibilizados pela nuvem, neste contexto, são o *Google Docs*[1], *Salesforce*[2].

1.3.3. Modelos de Implantação

Uma outra classificação bastante difundida baseia-se na forma que os ambientes de nuvens são implantados. De forma geral, usando este parâmetro, pode-se classificar as nuvens em privada (*private cloud*), pública (*public cloud*), comunitária (*community cloud*) e híbrida (*hybrid cloud*).

Nuvens privadas oferecem a ideia de fornecer serviços para a própria organização, sendo operadas e utilizadas apenas pela mesma. As nuvens privadas ainda podem ser classificadas em internas (locais) ou externas. O modelo de nuvem privada local, mostrado na Figura 1.4, permite que a organização tenha controle total sobre a infraestrutura e dados. Neste modelo, o departamento de TI da organização é tipicamente o prestador de serviços em nuvem. Este modelo é mais adequado para organizações que querem ter o controle completo sobre sua infraestrutura, configurações de recursos, aplicativos, dados e mecanismos de segurança.

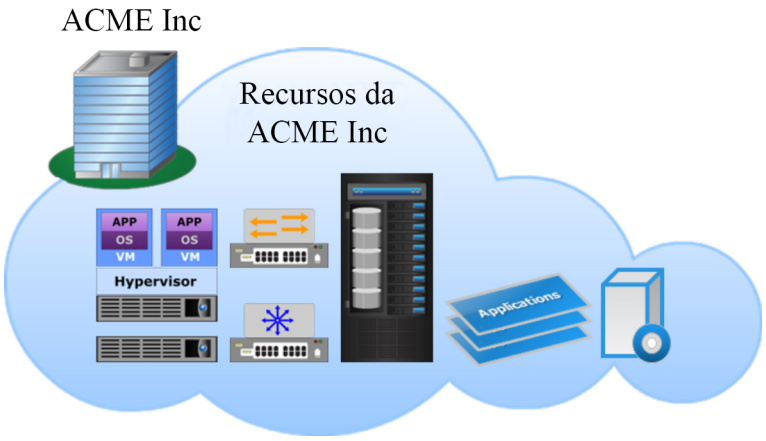


Figura 1.4: Nuvem privada interna

No modelo de nuvem privada hospedado externamente, mostrado na Figura 1.5, uma organização terceiriza a implementação da nuvem privada em um provedor de serviço de nuvem externa. A infraestrutura de nuvem está hospedada em instalações do provedor externo e não no interior das instalações da organização. O provedor gerencia a infraestrutura de nuvem e provê um ambiente privado exclusivo para a organização.

[1] `http://www.google.com/docs`
[2] `http://www.salesforce.com`

A infraestrutura de TI da organização se conecta à nuvem privada hospedada externamente através de uma rede segura. O provedor impõe mecanismos de segurança na nuvem privada por exigências de segurança da organização consumidora. Neste modelo, a infraestrutura de nuvem pode ser compartilhada por vários consumidores. No entanto, o provedor tem um perímetro de segurança em torno dos recursos de nuvem privada de cada uma das organizações consumidoras.

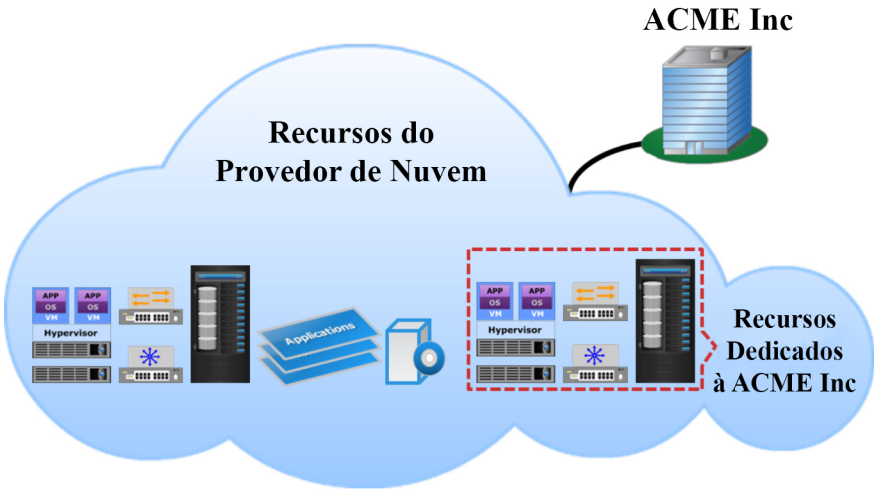


Figura 1.5: Nuvem privada hospeda externamente

As Nuvens Comunitárias baseiam-se um ambiente de Computação em Nuvem compartilhado entre organizações com interesses em comum.

Já as Nuvens Públicas comportam um modelo que disponibiliza ambientes para o público em geral e são normalmente comercializadas por corporações com grande poder de armazenamento e processamento.

E, finalmente, as Nuvens Híbridas tratam da composição entre dois ou mais ambientes de estruturas distintas, privadas e públicas, por exemplo, gerando uma única nuvem.

1.4. Plataformas para Computação em Nuvem

Uma plataforma para computação em nuvem pode ser definida como um sistema integrado para a criação e o gerenciamento de serviços em nuvens privadas, públicas e híbridas; combinado com a virtualização de servidores, armazenamento, rede e segurança através de uma abordagem centralizada para automatizar o ciclo de vida dos serviços na infraestrutura de nuvem.

Algumas plataformas de computação em nuvem estão disponíveis (VMware vRealize Suite, Openstack, OpenNebula, Cloudstack, Eucalyptus), porém nem todas são *open-source* e o custo com licenças pode exceder milhões de reais, dependendo do tamanho da infraestrutura.

As soluções *open-source* não fornecem suporte completo para todos os requisitos de usuário, como por exemplo o XenServer sem suporte a recuperação de desastre (DR) e o OpenNebula sem planejamento de capacidade.

O *VMware vCloud Suite* [14] permite a criação e gerenciamento de uma nuvem privada baseada no vSphere [5]. O funcionamento do vCloud é baseado na integração dos produtos VMware, criados para funcionarem melhor juntos, que fornecem: virtualização de infraestrutura, recuperação de desastres e automação de testes, e gerenciamento de nuvens privadas.

O principal obstáculo para adoção do VMware vCloud é o valor das licenças e o fato de não ser um software de código aberto, conforme recomenda o decreto 8.135 no que tange ao uso de software de código aberto. As instituições possuem requisitos específicos, muitas vezes não atendidos por produtos comerciais como o VMware, o que impacta fortemente a contratação dos serviços de nuvem.

O XenServer [13] é um projeto *opensource* gerenciado pela Citrix. É um projeto que desenvolve software de código aberto para virtualização de servidores x86, permitindo a consolidação de hardware e a automação para reduzir custos e simplificar o gerenciamento de servidores e aplicações.

Possuindo um foco mais voltado à virtualização de servidores e desktops, o XenServer consolidou-se como a segunda plataforma de virtualização mais adotado no mercado, principalmente quando se tornou *opensource* em 2013. Devido à esta decisão de foco da Citrix, o XenServer não atende à maioria dos requisitos definidos.

Openstack, Cloudstack e Eucalyptus concorrem entre si para se tornarem a plataforma padrão, sendo que o Openstack a mais promissoras de todas. Grandes empresas de tecnologia, como HP, IBM, RedHat, VMware, Cisco, Dell, EMC, Yahoo, etc.; fazem parte do consórcio que financiam e colaboram com o desenvolvimento do Openstack, muitas delas possuem uma versão licenciada com funcionalidades desenvolvidas e/ou adaptadas para seus produtos e soluções.

No seguimento de nuvem pública, o *Amazon Virtual Private Cloud* (VPC) [1] fornece uma instância logicamente isolado do *Amazon Web Services* (AWS) [2], onde é possível executar VMs em uma rede virtual definido por um usuário. É possível configurar o ambiente de rede virtual, incluindo a seleção de um intervalo de endereços IP, criação de sub-redes, e configuração de tabelas de rotas e roteadores de rede. Entretanto não fornece suporte à modelagem visual, apenas assistentes para facilitar a configuração.

1.5. Openstack

OpenStack é um projeto iniciado pela NASA e Rackspace. O objetivo inicial do projeto era criar uma alternativa open source compatível com *Amazon Elastic Compute Cloud* (EC2). Atualmente, OpenStack tem ganhado espaço como uma das principais opções para a industria de infraestrutura como serviço. Um ponto que torna OpenStack uma solução atrativa é sua arquitetura altamente modularizada. Com OpenStack, a implementação de um projeto de nuvem consiste na identificação das necessidades, especificação do hardware necessário e escolha adequada dos componentes necessários para atender aos requisitos da nuvem.

Uma tendência no mercado de infraestrutura é permitir que administradores e operadores entreguem uma infraestrutura completamente automatizada em minutos. Atualmente, muitas empresas como VMware, Cisco, Juniper, IBM, Red Hat, Rackspace, Pay-

Pal, and EBay, possuem nuvens privadas baseadas em OpenStack em seu ambiente de produção.

Para iniciar um projeto de criação de nuvem utilizando OpenStack, o primeiro passo é entender sua arquitetura e o funcionamento de cada um de seus componentes básicos.

1.5.1. Arquitetura e Componentes

Basicamente, os componentes do OpenStack podem ser divididos em 3 grupos:

- Controle
- Rede
- Computação

A camada de controle executa a API de serviços, interface web, banco de dados e serviço de mensagens. A camada de rede executa serviços de rede e agentes para gerenciamento de rede.Já a camada de computação provê serviços e agentes para tratar maquinas virtuais. Todos os componentes do OpenStack utilizam um banco de dados e/ou um serviço de mensagens.O banco de dados utilizado pode ser MySQL, MariaDB ou PostgreSQL. Os serviços de mensagens mais populares são RabbitMQ, Qpid, and ActiveMQ. Em ambientes menores, o banco de dados e serviços de mensagem normalmente executam no nó de controle, mas eles podem ter seus próprios nós caso necessário.

Em um ambiente com várias maquinas(nós), cada um desses grupos de componentes pode ser instalado em um servidor separado. O ambiente OpenStack pode ser instalado em uma ou duas maquinas, mas uma boa prática para garantir a escalabilidade é instalar cada um desses grupos de componentes em seu próprio servidor.

A tabela 1.1 apresenta de forma resumida os principais componentes que compõem o Openstack. Existem vários outros componentes em vários estágios de desenvolvimento que não são tratados nesse trabalho. Uma lista mais atualizada e completa desses componentes pode ser encontrada na página oficial do Openstack[3].

1.5.2. Painel de Controle: Horizon

A funcionalidade de gerenciamento do OpenStack é provida pelo Horizon Framework, que fornece as informações apresentadas na interface web provida pelo OpenStack. Horizon foi projetado para oferecer suporte de gerenciamento a todos os componentes oficialmente aceitos e inclusos no OpenStack. Basicamente a interface web do OpenStack é uma interface mais amigável que provê informações fornecidas pela API de gerenciamento do OpenStack. A Figura 1.6 apresenta uma das telas de gerenciamento do openstack.

1.5.3. Serviço de Identidade: Keystone

Keystone é o componente que provê uma API de autenticação, serviço de descoberta e autorização multi-tenant distribuída, através da implementação da API de identidade do OpenStack. Em sua instalação mais básica irá gerenciar tenants, usuários e perfis.

[3] www.openstack.org

Tabela 1.1: Componentes básicos OpenStack

Projeto	**Objetivo**	**Descrição**
Nova	Computação	Gerencia recursos de maquinas virtuais como CPU, memória, disco e interfaces de rede
Neutron	Rede	Fornece recursos utilizados pelas interfaces de rede das maquinas virtuais como endereçamento IP, roteamento e software defined networking (SDN).
Swift	Armazenamento de objetos	Provê o armazenamento de dados a nível de objeto (Object Storage) acessível via API REST.
Cinder	Armazenamento de blocos	Fornece armazenamento a nível de bloco (block storage) para máquinas virtuais
Keystone	Identidade/Autenticação	Gerencia o controle de acesso para os componentes do OpenStack. Provê serviços de autorização.
Glence	Serviço de Imagens	Gerencia imagens de maquinas virtuais. Provê entrega de imagens para maquinas virtuais e serviços de snapshot (backup de maquinas virtuais).
Horizon	Dashboard	Fornece uma interface web para gerenciamento da plataforma openstack.
Ceilometer	Telemetria	Fornece um conjunto de métricas para monitoramento dos componentes do OpenStack

Cada um dos componentes no OpenStack é registrado com um Keystone. Cada um dos serviços possui um *endpoint* e um usuário. Um serviço no Keystone é um registro de outro componente OpenStack que precisará ser contatado para gerenciar recursos virtuais. Endpoints são URLs utilizadas para acessar esses serviços. A imagem a seguir apresenta um a tela do Horizon que apresenta uma lista de serviços e seus respectivos endpoints para um determinado ambiente OpenStack.

1.5.4. Serviço de Computação: Nova

Nova é o componente de gerenciamento de instâncias. Nova executa um grande numero de requisições que colaboram para responder uma requisição do usuário final em uma maquina virtual. Um usuário autenticado com acesso ao Glence e uma rede configurada para as instâncias de maquinas virtuais no OpenStack já está apto a criar novas maquinas virtuais. Os últimos recursos necessários são um par de chaves e um grupo de segurança. O par de chaves utilizado pelo OpenStack é simplesmente um par de chaves SSH. É possível importar seu próprio par de chaves ou gerar um novo. Quando uma instância está executando, a chave pública é colocada no arquivo authorized_keys para que conexões SSH possam ser estabelecidas sem o uso de senha, de forma muito semelhante a outros serviços como Amazon EC2.

Antes que uma conexão SSH possa ser estabelecida, um grupo de segurança deverá ser configurado para permitir conexões externas. Um grupo de segurança (*security*

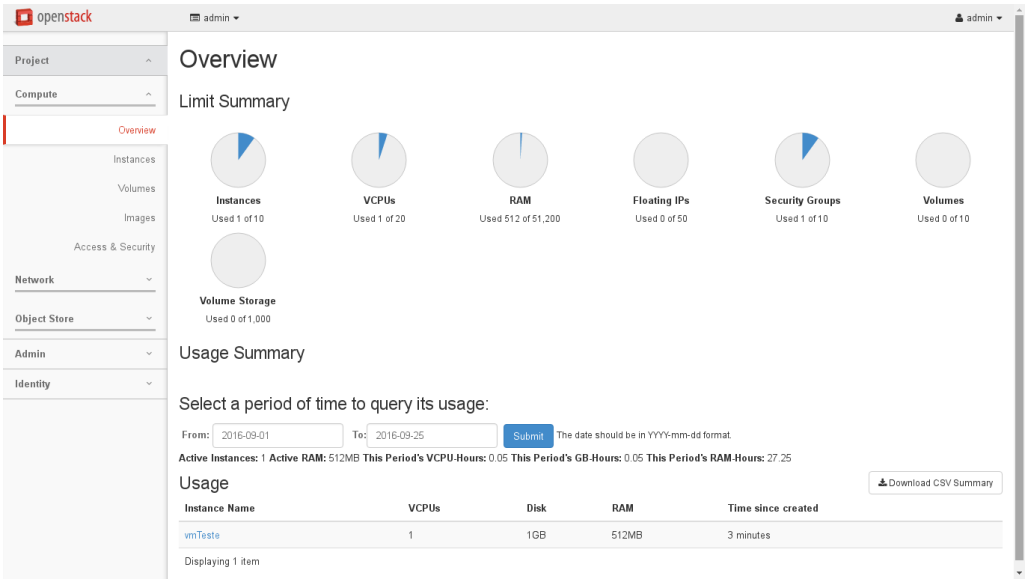


Figura 1.6: Interface Web Horizon

group) é um firewall para a infraestrutura de nuvem. Um grupo de segurança padrão é atribuido a instancias da nuvem para que elas possam se comunicar entre si, mas algumas regras como ICMP, SSH e outras conexões deverão ser criadas para permitir conexões fora do grupo de segurança.

Uma vez que haja uma imagem, rede, um par de chaves e grupos de segurança configurados, uma instância pode ser executada.

1.5.5. Serviço de Imagem: Glance

Glence é o componente de gerenciamento de imagens. Imagens de disco permitem que uma instância possa ser criada sem perda de tempo com instalação de sistema operacional e configuração de ambiente. Glence permite a criação e registro de imagens pré-configuradas que podem servir como base para a criação de novas instâncias de maquinas virtuais.

Imagens no Glence já vem com o sistema operacional instalado, mas possuem algumas informações como chave SSH do host e endereço MAC removidos. Isso faz com que a imagem possa ser copiada várias vezes durante a criação de novas instancias sem que haja conflito entre elas. Durante a inicialização de uma nova instância o informações específicas do host são geradas e atribuídas a essa nova instância.

1.5.6. Serviço de Rede: Neutron

Neutron é o componente para gerenciamento de rede, e gerencia infraestrutura de rede definida por software do OpenStack. Dentre outras coisas ele:

- Permite que o usuário crie sua própria rede e associe interfaces do servidor a ela
- Sua arquitetura baseada em plugins permite que usuários possam tirar vantagens de recursos disponíveis em dispositivos de rede específicos

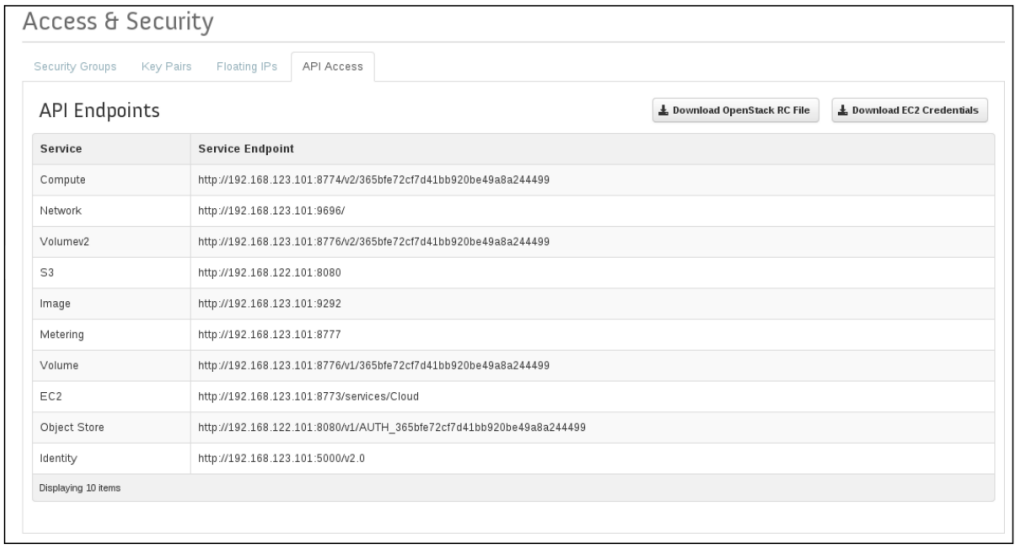


Figura 1.7: Serviços e Endpoints

- Extensões desse componentes permitem a integração com software, serviços de rede ou hardware

Neutron tem muitas funcionalidades importantes de rede que estão em constante crescimento e mudança. Algumas destas funcionalidades são úteis para roteadores, switches virtuais, e controladoras de rede SDN.

Qualquer rede configurada deve ter, pelo menos, uma rede externa. Ao contrário das outras redes, a rede externa não é apenas uma rede virtual. Representa uma abstração da rede física, sendo acessível por qualquer endereço IP fora do ambiente virtual que esteja ligado diretamente à rede física ou que possua rota definida.

Além de redes externas, qualquer rede configurada tem uma ou mais redes internas (sub-redes). Estas redes definidas por software conectam-se diretamente às VMs. Apenas as VMs da rede interna, ou aquelas em sub-redes conectadas através de um roteador, podem acessar VMs conectados diretamente a essa rede.

Os roteadores virtuais são necessários para conectar as VMs à rede exterior. Cada router tem uma porta de entrada ligado a uma rede externa e uma ou mais interfaces ligadas a redes internas. Assim como um roteador físico, sub-redes podem acessar máquinas em outras sub-redes que estão conectados ao mesmo roteador, e as máquinas podem acessar a rede externa através deste roteador. Para que uma VM seja acessa externamente é necessário alocar um endereço IP externo, ou endereço público.

1.5.7. Cinder

Cinder é o componente de armazenamento baseado em bloco. Volumes podem ser criados e associados a instâncias. Após isso, o volume pode ser usado da mesma forma que qualquer outro disco. O volume pode ser particionado, formatado e montado no sistema operacional. Cinder também permite a criação de snapshots. Snapshots podem ser gerados apartir de volumes(discos virtuais) ou instâncias de maquinas virtuais.

Existe uma grande variedade de tecnologias de armazenamento que podem ser configuradas para armazenar volumes e snapshots no Cinder. Por padrão, é utilizado LVM (Logical Volume Manager). É possível também utilizar tecnologias de armazenamento baseadas em software como GlusterFS e Ceph.

1.5.8. Swift

Swift é o componente de armazenamento baseado em objetos. Em um object storage os dados são armazenados como objetos. Um arquivo por exemplo, pode ser armazenado em um object storage apenas como conteúdo, sem os metadados associados a ele. A utilização do Swift como object storage traz algumas vantagens:

- A arquitetura descentralizada evita o surgimento de um ponto de falha único
- Possui mecanismos de autorecuperação em caso de falha
- Permite escalar o armazenamento para petabytes de dados através de técnicas de escalabilidade horizontal
- Melhor performance, que pode ser alcançada através da distribuição da carga de trabalho entre os nós de armazenamento
- Hardware de baixo custo pode ser utilizado para armazenamento de dados redundantes

1.6. Montandado uma Nuvem Privada com OpenStack

Nesta seção serão apresentados os detalhes que devem ser levados em consideração ao montar uma infraestrutura privada com o Openstack. Inicialmente será definido a configuração do ambiente, depois mostrados os passos de instalação, configuração, implantação de imagens e, por fim, uma introdução ao gerenciamento do nuvem privada.

1.6.1. Definição do Ambiente

Para instalar o Opemstack são necessárias 3 máquinas: uma para executar *Fronted* (horizon, keystone, cinder, glance e swift), outra para executar *Neutron* e uma para executar o *Nova Compute*. Podemos ter várias máquinas executando *Nova Compute*, aumentando assim a quantidade de recursos disponíveis e a quantidade de instâncias de máquinas virtuais que poderão ser executadas simultaneamente. É possível executar tudo em apenas uma única máquina, porém seria necessário ter uma especificação de hardware bem poderosa para prover todos os recursos necessários pelos componentes, sendo que este cenário não é recomendo para ambientes de produção. A Figura 1.8 mostrada o ambiente mínimo recomendado para execução do Openstack.

Antes de instalar e usar o Openstack, é necessário avaliar os requisitos de hardware. A Tabela 1.2 mostra especificações mínimas recomendadas para máquinas que executarão o *Frontend* e *Neutron*.

As máquinas que executarão o serviço de *Nova Compute* precisam ter uma configuração mais potente, pois eles irão executar instâncias das máquinas virtuais. A Tabela 1.3 mostra alguns exemplos

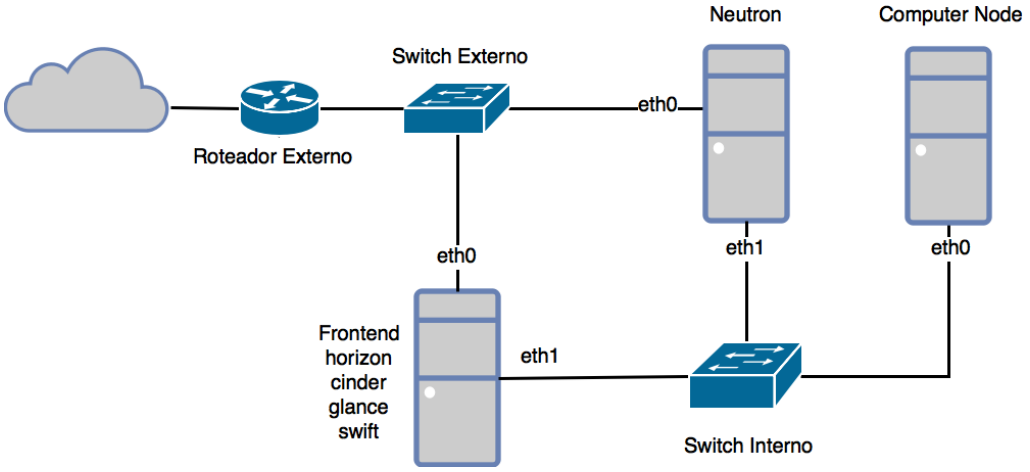


Figura 1.8: Ambiente mínimo para instalação do Openstack

Tabela 1.2: Requisitos de Hardware para Frontend

Hardware	**Mínimo**	**Sugerido**
CPU	1 GHz	2 x 2 GHz
Memória	1 GB	8 GB
Disco	SATA 7200 rpm	SAS 15000 rpm ou SSD
Espaço em Disco	500 GB	4 TB
Rede	1 Gbps	10 Gbps

Em um ambiente de produção a quantidade de servidores utilizados depende diretamente da quantidade de máquinas virtuais que podem executar simultaneamente e dos recursos alocados em cada máquina virtual (VM), como memória RAM e espaço em disco.

Suponde que o ambiente onde vai ser implantado possua as seguintes características:

- 300 VMs
- Quantidade máxima de vCPU por VM: 2
- Quantidade máxima de RAM por VM: 8
- Processador do servidores: Intel Xeon Processor E5-2698 v4 2.2 Ghz 10 Cores
- Quantidade de (socket) processador por servidor: 2
- Quantidade de *threads* por processador: 2

Em cada servidor executaríamos no máximo 20 VMs, calculado pela fórmula:

$$SOCKET * CORES * THREADS / MAX\_vCPU \Rightarrow (2 * 10 * 2)/2 = 20$$

Tabela 1.3: Nova Compute

Hardware	**Mínimo**	**Sugerido**
CPU	Extensões VT/VT-X/AMD-v	VT/VT-X/AMD-v, 64 bits, vários núcleos
Memória	4 GB	16 GB
Disco	IDE 5400 rpm	SATA, SCSI ou SAS 7200/10000 rpm
Espaço em Disco	100 GB	500 GB
Rede	1 Gbps	10 Gbps

Precisaríamos de 600 cores para executar 300 VMs, calculado pela fórmula:

$$TOTAL_VMS * MAX\_vCPU \Rightarrow 300 * 3 = 600$$

Precisaríamos de 15 para servidores, calculado pela fórmula:

$$TOTAL\_CORES / CORES\_POR\_SERVIDOR \Rightarrow 600/40 = 15$$

Precisaríamos de 192 GB de RAM por servidores, calculado pela fórmula:

$$VMS\_POR\_SERVIDOR * RAM\_POR\_VM \Rightarrow 20 * 8 = 160 \sim 192$$

Porém a real carga e quantidade de máquinas virtuais por servidor dependem da carga que vai ser executada, podendo variar para mais ou para menos.

1.6.2. Instalação e Configuração

O Openstack pode ser instalado a partir da compilação do código fonte ou através de pacotes específicos das distribuições Linux. Os pacotes prontos para instalação estão disponíveis para Redhat através do gerenciador de pacotes YUM e distribuições baseadas em Debian através do gerenciador de pacotes APT.

A instalação e configuração manual do OpenStack envolve a instalar, configurar e registrar cada um dos componentes, assim como seus vários bancos de dados e um sistema de mensagens. É muito complicado, repetitivo, e suscetível à erros, além do processo ser um pouco confuso. Felizmente, existem algumas distribuições que incluem ferramentas para automatizar este processo de instalação e configuração.

O RDO [4] é uma distribuição suportada pela comunidade do Red Hat Openstack que automatiza a instalação e configuração do Openstack em sistemas baseados em pacotes RPM. O RDO utiliza os pacotes oficiais do Openstack, sem modificações, para as diversas versões do Openstack: Kilo, Liberty, Mitaka (versão estável mais atual). o RDO será utilizado como ferramenta para demonstração da instalação do Openstack.

Para fins de demonstração utilizaremos uma única máquina para rodar todos os serviços Openstack. A instalação será executada em um servidor Linux CentOS 7 x64.

Depois de ter executado a instalação do CentOS 7 no servidor (você pode usar máquina virtual para esta finalidade também), certifique-se ter configurado um o IP estático

[4] https://www.rdoproject.org/

durante a instalação para seu sistema; caso contrário, o DHCP deverá ser desativado para a interface de rede e configurar um endereço IP estático. Após configurar o IP estático, desativar o serviço NetworkManager, pois o Neutron pode apresentar problemas com o gerenciamento de interfaces.

```
systemctl stop NetworkManager.service
systemctl disable NetworkManager.service
systemctl restart network
```

Desativar temporariamente o SELinux para evitar quaisquer erros estranhos durante a instalação e parar firewalld (ou adequadamente configurá-lo para permitir todas as portas necessárias, etc.).

```
setenforce 0
systemctl stop firewalld
systemctl disable firewalld
```

O primeiro passo é a instalação do pacote do RDO utilizando o comando yum:

```
yum install https://rdoproject.org/repos/rdo-release.rpm -y
```

Execute seguinte comando para instalar a versão da estável do OpenStack para CentOS. Observando a que a versão utilizada do pacote é a 1-2.el7:

```
yum install centos-release-openstack-mitaka-1-2.el7 -y
```

Execute seguinte comando para instalar utilitário Packstack:

```
yum install openstack-packstack -y
```

Assim que a instalação Packstack estiver concluída, execute seguinte comando para instalar o OpenStack Mitaka:

```
packstack −−allinone −−provision-demo=n −−os-neutron-lbaas-install=y
−−neutron-fwaas=y −−os-neutron-vpnaas-install=y −−os-neutron-ovs-bridge-
mappings=extnet:br-ex −−os-neutron-ovs-bridge-interfaces=br-ex:eth0
```

O Packstack possui vários parâmetros para configuração, para object uma lista detalhada deve-se executar o comando *packstack −−help*. As opções utilizadas foram as seguintes:

- −−allinone: instalação de todos os componentes Openstack no mesmo servidor;
- −−provision-demo=n: ignorar a criação de usuário demo

- −−os-neutron-lbaas-install=y: instalar o serviço de *Load Balancer* como Serviço, pois o Packstack não instala por padrão;
- −−neutron-fwaas=y: instalar o serviço de *Firewall* como Serviço, pois o Packstack não instala por padrão;
- −−os-neutron-vpnaas-install=y: instalar o serviço de *VPN* como Serviço, pois o Packstack não instala por padrão;
- −−os-neutron-ovs-bridge-mappings=extnet:br-ex: Nome da *bridge* para a rede externa que irá ser utilizada pelo Neutron;
- −−os-neutron-ovs-bridge-interfaces=br-ex:eth0: Interface que rede que irá fazer parde ta *bridge* externa do Neutron.

A execução do comando vai demorar vários minutos para concluir a instalação, verificar nos logs o andamento do processo. A Figura 1.9 mostra o que você deve ver no caso de instalação bem-sucedida. Ela vai mostrar a URL de administração do *Horizon*.

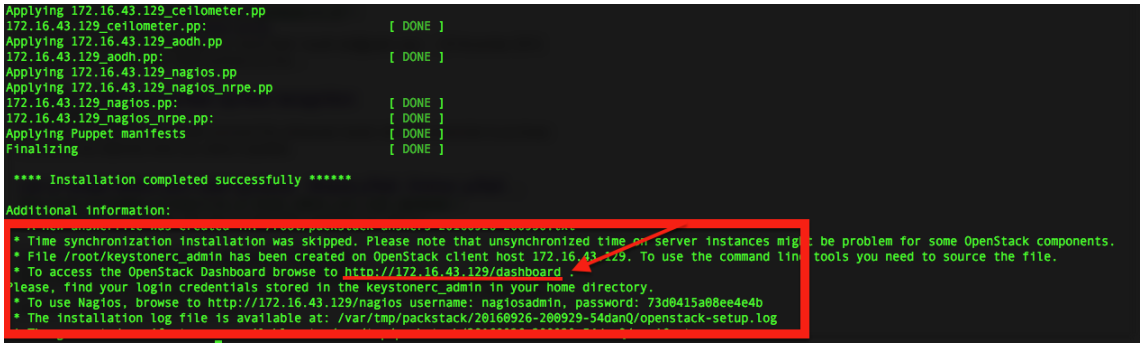


Figura 1.9: Execução Packstack

Você será capaz de localizar as informações de login do OpenStack dentro do arquivo "/root/keystonerc_admin"como indicado na Figura 1.10. Execute o comando abaixo para ver o conteúdo do arquivo:

```
cat /root/keystonerc_admin
```

Após obter as credencias de admin no arquivo keystonerc_admin, deve-se abrir a página de gerenciamento do Openstack em um navegador web utilizando as credenciais, conforme a Figura 1.11.

1.6.3. Gerenciamento de Usuários

Tudo no OpenStack deve existir dentro de um *tenant*. Um *tenant* é simplesmente um agrupamento de objectos. Usuários, VMs e redes são exemplos de objetos. Eles não podem existir fora de um *tenant*. Outro nome para um *tenant* é projeto. Na linha de comando (CLI - Command Line Interface), o termo *tenant* é utilizado. Na interface Web (*Horizon*), o termo projeto é usado.

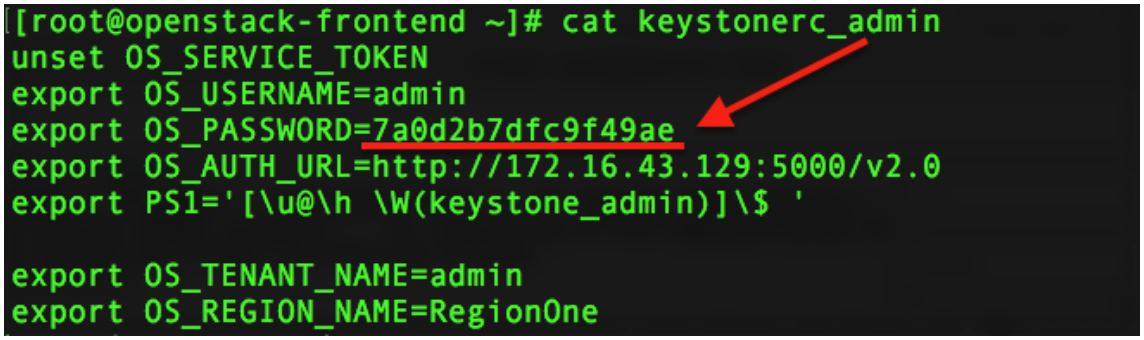


Figura 1.10: Credenciais do Openstack

A Figura 1.12 mostra a criação de um project (*tenant*) através do *Horizon*. Menu ***Identity -> Projects***, botão ***Create Project***.

Durante a criação de um usuário através da CLI deve-se especificar o nome do usuário, a senha e em qual *tenant (project)* ele estará ligado, para isso utilizamos o seguinte comando:

```
keystone user-create −−name=demo1 −−pass=123 −−tenant=demo
```

1.6.4. Registro de Imagens no Glance

Antes de criar máquinas virtuais faz-se necessário o registro das imagens a serem utilizadas no Glance. Na documentação do Openstack é indicado o link de imagens homologadas de diversos sistemas operacionais [5].

Para fins de demonstração a imagem do CirrOS[6] Linux será utilizada, por tratar-se de uma mini distribuição Linux específica para testes em plataforma de nuvem.

O registro de imagens pode ser feito via CLI ou via interface gráfico no Horizon. Via CLI, deve-se carregador as variáveis de ambiente presente no arquivo keystonerc_admin. Para isso executarmos o seguinte comando:

```
source /root/keystonerc_admin
```

Depois deve-se fazer o download das imagens:

```
wget http://download.cirros-cloud.net/0.3.3/cirros-0.3.3-x86_64-disk.img
```

Finalmente o registro da imagem no Glance:

```
glance image-create −−name "cirros"−−file cirros-0.3.3-x86_64-disk.img −−disk-format qcow2 −−container-format bare −−visibility public −−progress
```

[5] `http://docs.openstack.org/image-guide/obtain-images.html`
[6] `http://download.cirros-cloud.net/0.3.4/cirros-0.3.4-x86_64-disk.img`

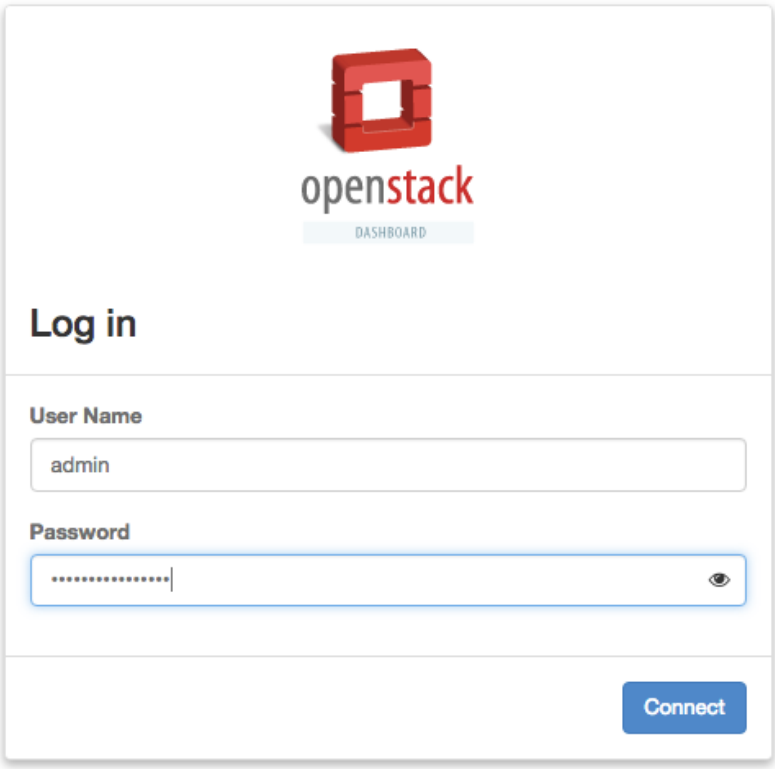


Figura 1.11: Tela de Login do Horizon

1.6.5. Criação da infraestrutura de rede virtual

Antes de iniciar uma instância, você deve criar a infra-estrutura de rede virtual necessária. Faz-se necessário e obrigatório a existência de uma rede pública (externa) para que as VMs conseguiam acessar a Internet ou qualquer rede fora do ambiente virtual do Openstack. Logado como administrador executar o seguinte comando para criar a rede:

```
neutron net-create public −−router:external=True
```

Onde o parâmetro *"−−router:external=True"* indica ao Neutron que a rede criado é do tipo externa.

Após criar a rede externa, deve-se criar a sub-rede:

```
neutron subnet-create public 172.16.43.0/24 −−name public_subnet
−−enable_dhcp=False −−allocation_pool start=172.16.43.100,end=172.16.43.250
−−gateway 172.16.43.2
```

Na definição da sub-rede devemos informar o intervalo de endereço que poderão ser atribuídos às VMs em caso de acesso da rede física para a rede virtual, assim como o endereço do roteador padrão da rede física.

Cada usuário do Openstack pode ter sua própria definição de rede isolada dos

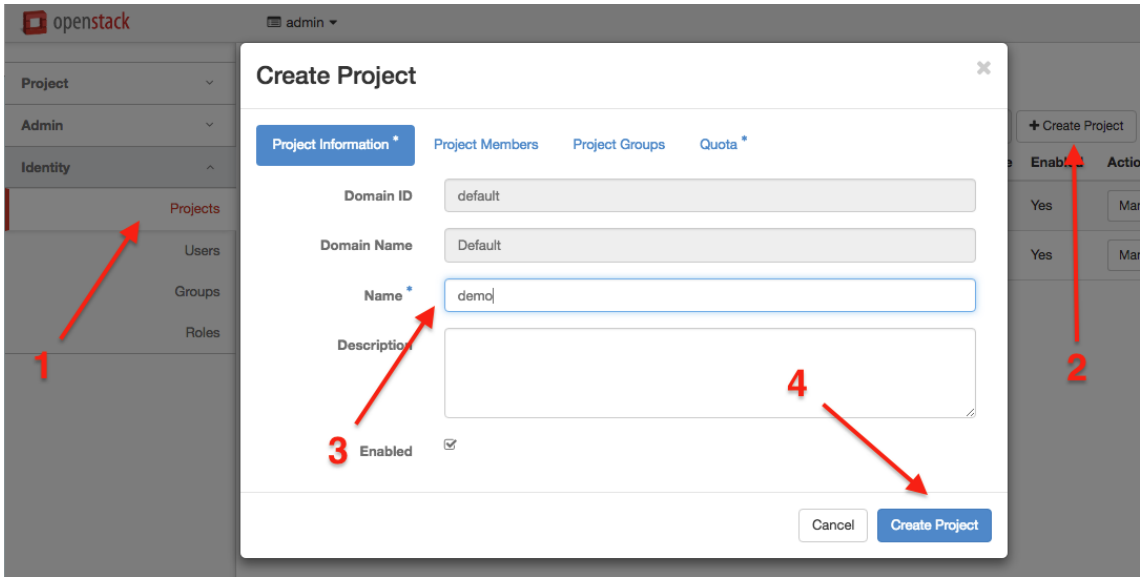


Figura 1.12: Criação de Projeto

demais. A infraestrutura de rede virtual do cliente, basicamente, consiste em uma rede privada, que encapsula uma sub-rede com uma range de endereço IP associadas sendo fornecido pelo serviço de DHCP do **Neutron** e um roteador ligado à um rede externa (criada previamente pelo administrador).

Para isso descobrir qual o ID do *tenant* (projeto) para o qual que iremos criar as definições de rede:

```
keystone tenant-list
```

```
+----------------------------------+----------+---------+
|                id                |   name   | enabled |
+----------------------------------+----------+---------+
| bbf183a282e3459f90f0a5cede56f058 |  admin   |   True  |
| b7cac0ea30d1460c84096f7b6c005e20 |   demo   |   True  |
| 48dea2ecc3e5468cafae33252dacf694 | services |   True  |
+----------------------------------+----------+---------+
```

Figura 1.13: Listagem de Tenants

De posso do ID do *tenant* executar os seguintes comandos:

```
neutron net-create private –tenant-id b7cac0ea30d1460c84096f7b6c005e20
```

O parâmetro *"––tenant-id"* indica para qual *tenant* será criado a rede.

Para criação da sub-rede privada:

```
neutron subnet-create private 192.168.69.0/24 −−name subnet-private
−−enable_dhcp=True −−allocation-pool start=192.168.69.100,end=192.168.69.200
−−dns-nameserver=8.8.8.8 −−tenant-id b7cac0ea30d1460c84096f7b6c005e20
```

Onde **“192.168.69.0/24”** é o endereço da rede privada do *tenant*, podendo ser qualquer endereço privado da escolha do usuário (10.0.0.0/8, 172.16.0.0/12 ou 192.168.0.0/16). Observando que outro *tenant* pode criar uma sub-rede com o mesmo endereço, pois cada sub-rede é isolada e atrelado ao usuário que criou a rede, não interferindo nas definições dos demais usuários.

Para definição e configuração do serviço de DHCP para esta rede usa-se o parâmetro: **“−−allocation-pool”**, definindo o início e fim da alocação. Também é possível informad qual o endereço do servidor DNS que será distribuído pelo DHCP com o parâmetro **“−−dns-nameserver”**.

```
neutron router-create router-demo −−tenant-id b7cac0ea30d1460c84096f7b6c005e20
neutron router-interface-add router-demo subnet-private
neutron router-gateway-set router-demo public
```

A criação de um roteador para ligar a a rede privada na rede externa faz-se com o comando: **neutron router-create**, depois deve-se adicionar uma interface da sub-rede do *tenant* e definir que a padrão será pela rede externa.

A Figrua 1.14 mostra a topologia da rede criado para o *tenant* demo, onde todos os usuário deste projeto poderão utilizar para conectar VMs.

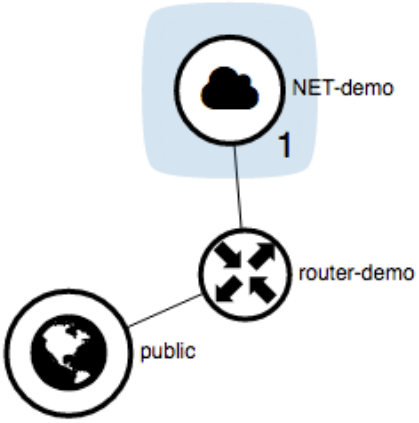


Figura 1.14: Topologia da Rede Virtual

1.6.6. Inicialização de instâncias

Após todo o ambiente necessário para a inicialização de instâncias ser configura, finalmente pode-se criar novas instâncias de VMs.

Via CLI, caso exista apenas uma sub-rede criado para o *tenant*, executa-se o seguinte comando:

```
nova boot --image=cirros --flavor=m1.tiny vm1
```

Onde **"--image"** defini-se a partir de qual imagem será criada a VM e **"--flavor"** qual configuração de CPU, memória e disco será utilizada.

Para a criação de VMs com mais detalhes, deve-se utilizar o *Horizon*, conforme mostrado pelas figuras a seguir:

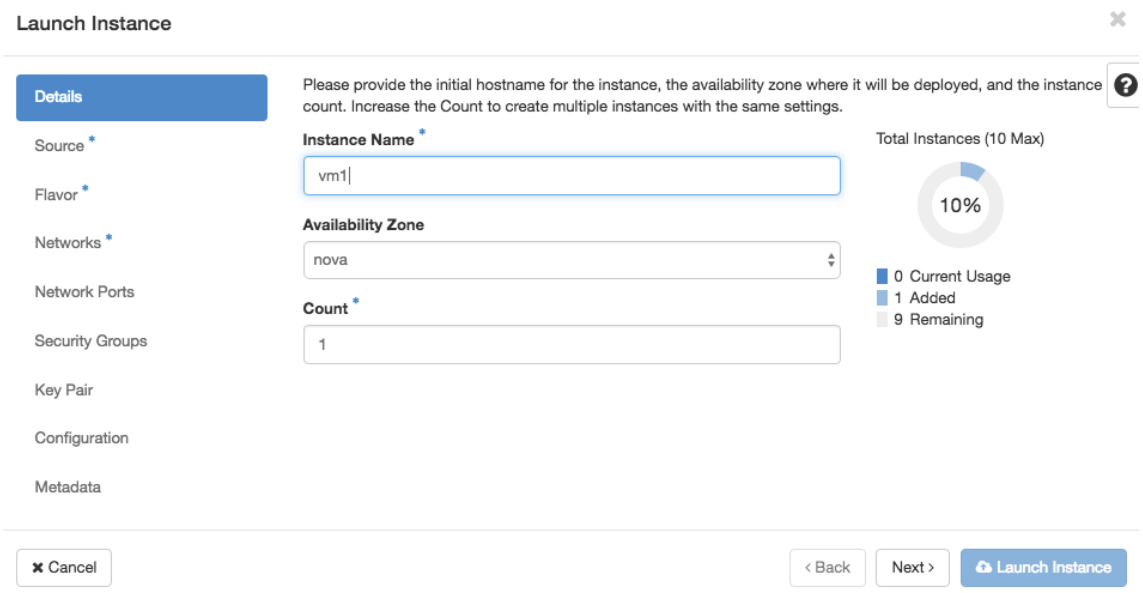


Figura 1.15: Criação de VM: Definição de Informações Báscias

1.7. Considerações Finais

O uso do Openstack para construir uma infraestrutura privada em um ambiente de computação em nuvem tem-se mostrado uma solução promissora, mesmo sendo ferramenta Opensource, possui o apoio e patrocínio de grandes empresas de hardware e software.

Este minicurso apresentou uma visão geral de seus componentes, mostrando um passo-a-passo deste a definição do ambiente até criação e execução de máquinas de máquinas virtuais.

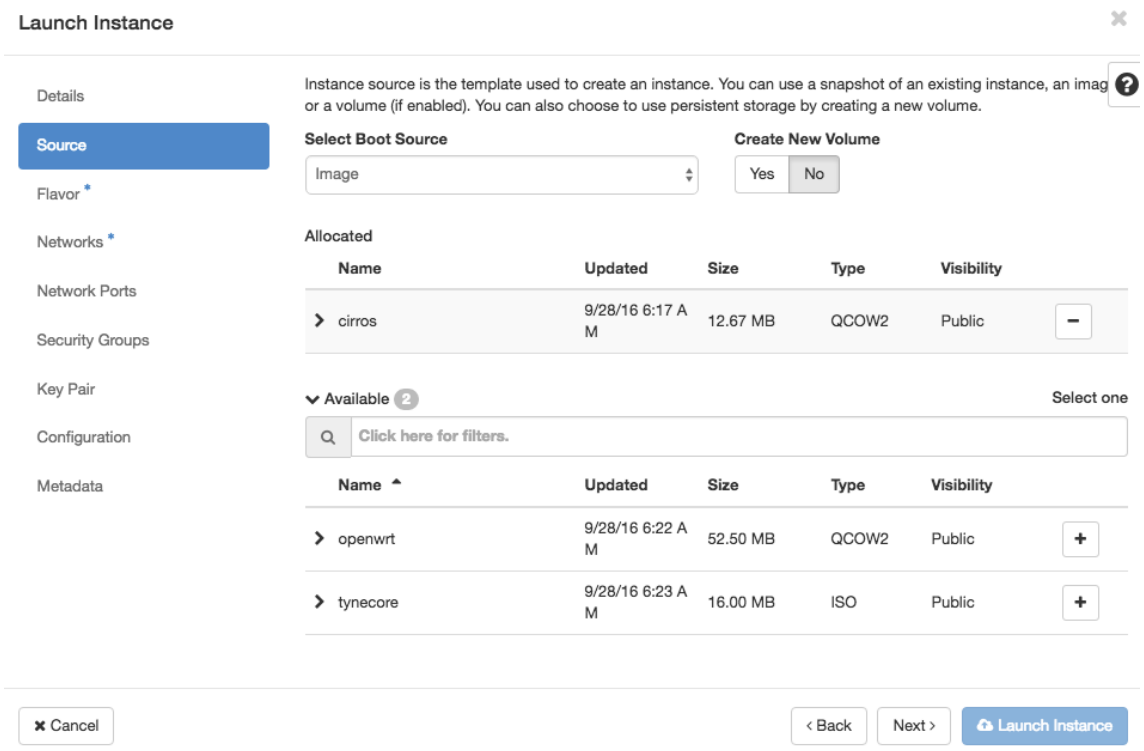


Figura 1.16: Criação de VM: Definição do Imagem

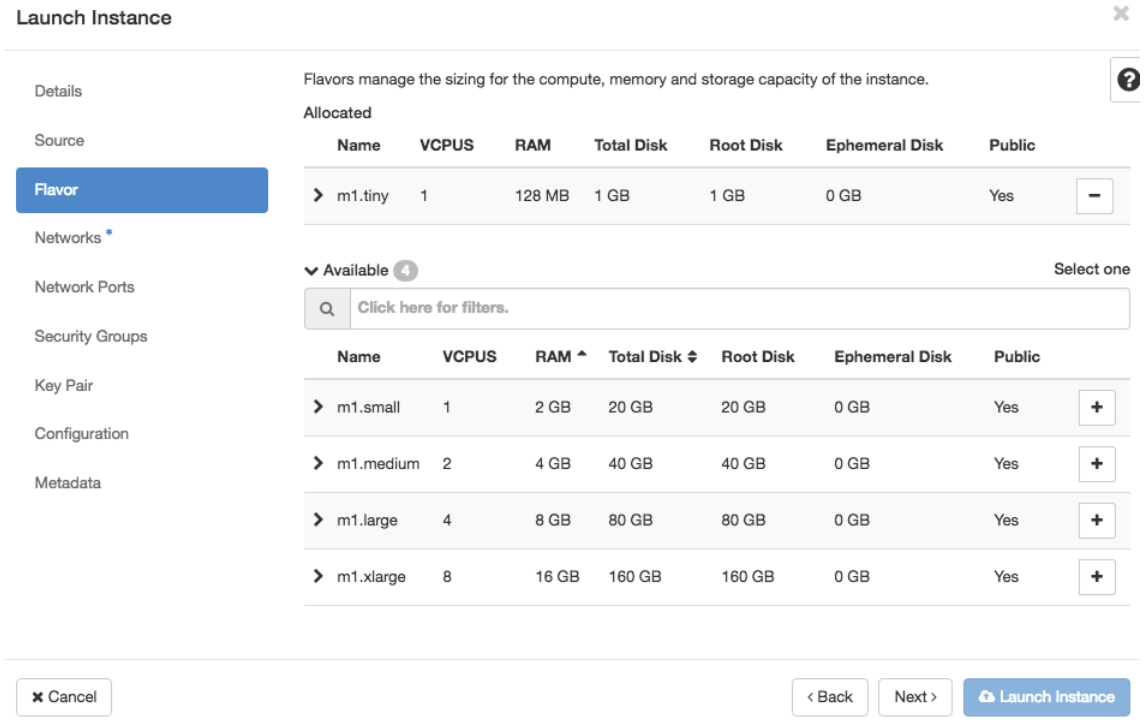


Figura 1.17: Criação de VM: Definição do Flavor

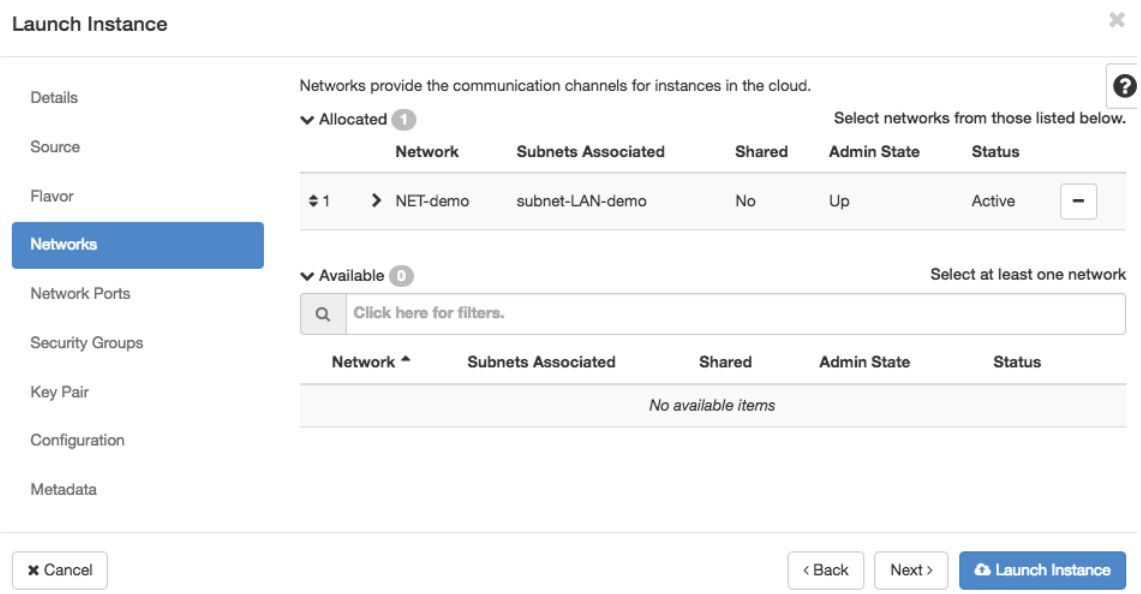


Figura 1.18: Criação de VM: Definição da Rede

Referências

[1] Amazon Web Services. Amazon virtual private cloud: User guide. http://awsdocs.s3.amazonaws.com/VPC/latest/vpc-ug.pdf/, 2014.

[2] Amazon Web Services, Inc. Amazon Web Services. http://aws.amazon.com/, 2015.

[3] Alexandre Carissimi. Virtualizacao: da teoria a solucoes. *Minicursos do Simposio Brasileiro de Redes de Computadores - SBRC 2008*, pages 173–207, 2008.

[4] Frederic and Magoules. *Introduction to Grid Computing (Chapman & Hall/CRC Numerical Analysis and Scientific Computing Series)*. CRC Press, 5 2012.

[5] VMware Inc. VMware vSphere. http://www.vmware.com/products/vsphere, 2015. [Online; accessed 15-Jan-2015].

[6] Neil MacDonald. Yes, hypervisors are vulnerable. http://blogs.gartner.com/neil_macdonald/2011/01/26/yes-hypervisors-are-vulnerable/, 2011.

[7] Maciej Malawski, Maciej Kuzniar, Piotr Wojcik, and Marian Bubak. How to Use Google App Engine for Free Computing. *IEEE Internet Computing*, 17(1):50–59, January 2013.

[8] Peter Mell and Tim Grance. SP 800-145. The NIST Definition of Cloud Computing. *National Institute of Standards and Technology*, 2011.

[9] Daniel A. Menasce. Virtualization: Concepts, Applications, and Performance Modeling. *Int. CMG Conference*, pages 407–414, 2005.

[10] Dejan Milojicic, Ignacio M. Llorente, and Ruben S. Montero. OpenNebula: A Cloud Management Tool. *IEEE Internet Computing*, 15(2):11–14, March 2011.

[11] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The Eucalyptus Open-Source Cloud-Computing System. In *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 124–131. IEEE, May 2009.

[12] Robert Rose. Survey of system virtualization techniques. http://ir.library.oregonstate.edu/xmlui/handle/1957/9907, 2004.

[13] N. Staalinprasannah and S. Suriya. Implementation of Xenserver to Ensuring Business Continuity Through Power of Virtualization for Cloud Computing. In *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, pages 1–6. IEEE, July 2013.

[14] Inc VMware. VMware vCloud Suite. http://www.vmware.com/products/vcloud-suite, 2015.

Capítulo 2

Implementação de jogos First Person Shooter utilizando Oculus Rift e Unity3D

Ruhan Bello

Abstract

This text is relative to a short course for ENUCOMP 2016 about the implementation of First Person Shooter games using the Unity3D game engine and the Oculus Rift device. This text aims to explore the basic concepts about Unity Editor, C# programming for Unity3D and the basic step-by-step to add the Oculus Rift on a Unity3D Project.

Resumo

Este texto é relativo a um minicurso oferecido no ENUCOMP 2016 sobre a implementação de jogos First Person Shooter utilizando a game engine Unity3D e o dispositivo Oculus Rift. Este texto tem como objetivo explorar os conceitos básicos sobre o Unity Editor, programação em C# para Unity3D e o passo-a-passo básico para a adição do Oculus Rift em um projeto na Unity3D.

2.1. Introdução

O mundo dos jogos tem passado por diversas transformações com o passar dos anos. Diversas questões surgiram em todas as áreas da criação de jogos, sobre como deixar um jogo mais realista, mais divertido, mais interativo e muito foi aprendido com essas questões.

Uma das características mais importantes, que está presente em todas as mídias interativas é a imersão, que faz com que uma determinada pessoa acredite que está de fato vivenciando uma experiência fora da realidade. Cinema, quadrinhos, jogos, música, todas estas mídias têm o poder de causar imersão. A partir disto, surgem questões importantíssimas como “Como posso tornar o meu jogo mais imersivo?”

Foi graças a este tipo de questionamento que ocorreu a criação de equipamentos que bloqueiam a visão que o usuário tem do mundo real, e o limitam a ver apenas o

mundo virtual através de uma tela, fazendo assim com que seja mais fácil que um jogador acredite que ele está lá participando dos eventos que ocorrem dentro do mundo virtual e fora do mundo real e, dessa maneira, aumentando drasticamente a imersão do usuário. Isso é o que chamamos de **Realidade Virtual**.

Um destes equipamentos que foi criado é um dos principais objetivos de estudo deste minicurso: o ***Oculus Rift***, uma tecnologia que evoluiu bastante até chegar em um ponto bastante aceitável tanto para jogadores, quanto para desenvolvedores. Mas existem algumas dúvidas de desenvolvedores que querem se aventurar com a Realidade Virtual: o quão difícil é desenvolver uma aplicação utilizando o *Oculus Rift*? Afinal de contas, os conceitos que envolvem a criação do equipamento não são os mais simples. São "tecnologia de ponta".

Responderemos a esta pergunta através do principal objetivo deste minicurso, que é explorar os conceitos da **Unity3D**, uma das mais populares plataformas de desenvolvimento de jogos, buscando os conhecimentos mais básicos que um desenvolvedor necessita para se iniciar neste mundo e partindo, por fim, a um breve estudo do *Rift.*

2.2. *Game Engine* Unity3D

Game Engines são *frameworks* criados especificamente para auxiliar na criação de jogos. Ou seja, em sua grande maioria vêm com conjuntos de ferramentas para auxiliar o desenvolvedor em tópicos como renderização, cálculos de física, iluminação, emissão de som, etc.

Isto tem se tornado popular na medida em que o mundo todo se vê mais direcionado à produção de jogos. Com o aumento do número de pessoas começando a fazer jogos, temos também o aumento do número de iniciantes nessa área. Graças a isto, um dos principais aspectos que as *Game Engines* incorporam é o balanço entre: facilidade de desenvolvimento e as possibilidades de criar projetos complexos.

Neste texto, iremos analisar o uso da Unity3D, que se tornou bastante popular nos últimos anos justamente por fazer um ótimo balanço das características citadas acima, prova disso é o fato de já existirem inúmeros jogos simples sendo feitos por iniciantes, assim como temos inúmeros jogos mais complexos sendo feitos até mesmo por empresas grandes, tal como o jogo ***Grow Up***, da **Ubisoft**.

2.2.1 Unity Editor

Muitas vezes a simplicidade do processo de desenvolvimento de qualquer *software* através de um *framework* está presente em ferramentas que possibilitam desenvolver de maneira visual. Esta é a principal função do *Unity Editor*: proporcionar ao desenvolvedor maneiras de criar o mundo do seu jogo, informando tudo o que estará presente neste mundo e como cada coisa nele se comporta. Na Unity, boa parte disso é feito de maneira visual.

Neste texto, primeiro iremos analisar o editor e logo em seguida iremos criar, do zero, o nosso projeto de *First Person Shooter* para explorar os detalhes que envolvem o projeto. Para explorar tais detalhes, iremos desenvolver o jogo passo-a-passo com instruções e explicações acerca de cada passo.

Se você já possui os conhecimentos básicos sobre o editor, sinta-se livre para continuar a leitura deste texto a partir do tópico **2.2.2.**

2.2.1.1 Aba *Project*

Nesta aba é onde estarão localizados os arquivos do projeto em desenvolvimento. Aqui serão armazenados todo e qualquer conteúdo necessário para a criação do jogo, que não estejam presentes por default no próprio editor. Como pode-se observar na Figura 2.1, a pasta raiz de todo projeto, por padrão, é a pasta *Assets*.

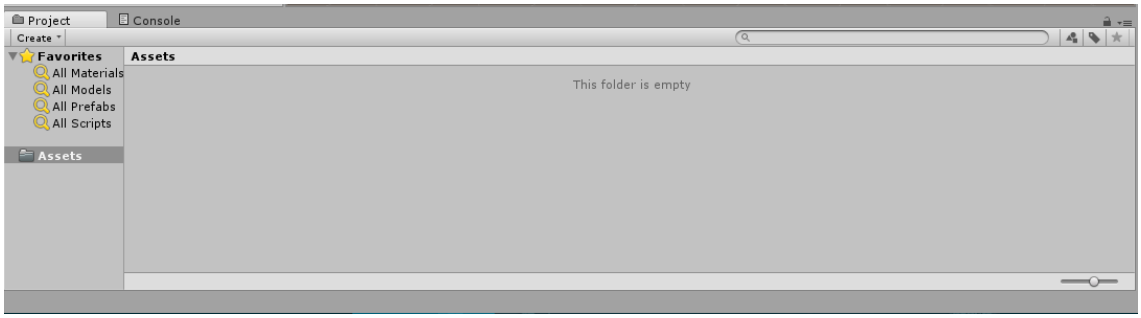


Figura 2.1. Aba *Project*

A partir daí, fica a critério do próprio desenvolvedor criar e manipular sua estrutura de pastas de acordo com os seus próprios padrões de projeto.

Para criar novas pastas ou até mesmo novos *assets*, basta clicar com o botão direito do mouse em qualquer campo vazio e acessar o sub-menu *Create*.

Deve-se ressaltar que os *assets* podem ser tanto criados a partir do próprio editor, e portanto será criado com as caractesíticas mais simples possíveis, ou estes podem ser importados de fora da Unity, possibilitando que o desenvolvedor utilize *assets* que já possuem alguma configuração.

Um exemplo de *asset* criado dentro do próprio editor é um material a ser aplicado em um modelo 3D. Quando o desenvolvedor o cria dentro da própria Unity, este material terá obrigatoriamente as configurações mais básicas de um material. Porém, caso o desenvolvedor deseje um material que venha já venha com alguma configuração específica, ele pode simplesmente importar de fora da Unity.

Para importar assets, deve-se clicar com o botão direito do mouse em alguma das pastas da aba *Project* e clicar no item *Import New Asset...*

Outra possibilidade interessante é a de importar ou exportar pacotes, acessando o menu com o botão direito e clicando em *Import/Export Package*. A vantagem disso é que as vezes o desenvolvedor pode necessitar importar um conjunto de *assets* de uma só vez, contidos em um pacote. Portanto, pacotes podem possuir poucos *assets*, como por exemplo dois materiais, mas podem possuir também vários *assets*, como por exemplo uma cena de jogo inteira composta por modelos 3D de personagens e cenários, os materiais e texturas destes modelos, scripts, etc. Neste texto nós iremos utilizar alguns pacotes que contém cenários, personagens, etc, pois o foco deste texto não é a criação de modelos 3D e nem texturização, portanto nós iremos reutilizar os que já existem.

É possível importar pacotes que não são disponibilizados pela própria Unity, clicando no item *Custom Package*.

Uma maneira alternativa de importar *assets* pro projeto é através da ***Asset Store***, uma loja virtual da Unity, mas falaremos sobre essa loja posteriormente neste texto.

2.2.1.2 Aba *Scene*

Esta aba é responsável por dar uma visão ao desenvolvedor de como estão posicionados os objetos no seu mundo virtual e possibilita que o jogador navegue por este mundo para observá-lo de qualquer ângulo e posição. Através desta aba também é possível editar três características dos objetos: posição, rotação e escala (tamanho). Estas três características são guardadas no atributo ***Transform***.

Existem diversos tipos de objetos em um mundo virtual na Unity3D: modelos 3D, câmeras, fontes de luz, fontes de som, etc, e é importante saber que o atributo mínimo que todo objeto possui é o atributo *Transform*.

Nesta aba estão presentes alguns botões e parte destes serão explicados a seguir e poderão ser observados na Figura 2.2

É possível modificar como os objetos serão renderizados na tela expandindo o sub-menu *Shading Mode*.

O botão 2D serve para alternar a maneira de visualização entre 2D e 3D. A diferença é que quando a cena está sendo visualizada no modo 2D, o eixo Z será ignorado e tudo será visto de maneira “chapada”, ou seja, como se tudo estivesse no mesmo plano e não houvesse profundidade entre os objetos da cena. Isto é bastante útil na hora de criar projetos 2D, onde o eixo Z muitas vezes é desnecessário.

O botão que mostra o ícone de um sol faz com que a cena seja vista ignorando ou não a iluminação presente nesta.

O botão que mostra o ícone de um alto-falante serve para silenciar ou não os áudios que houverem na cena.

O botão que mostra o ícone de uma foto serve para definir o que vai ser renderizado na aba *Scene*.

O último detalhe sobre aba *Scene* é um conjunto de cones apontando para um cubo, que chamamos de *gizmo*. No gizmo, os cones são os eixos X, Y e Z e ajudam quando o desenvolvedor deseja saber sob qual ângulo ele está visualizando a sua cena e caso clique em algum dos ícones, ele forçará a visão a ir para aquela direção. Ou seja, se o desenvolvedor clicar no cone que tem um X escrito, ele verá a cena como se ele estivesse no canto direito desta. Caso clique no cubo no centro destes cones, ele irá mudar entre os tipos **isométrico** (ou **ortográfico**) e **perspectiva**. O modo perspectiva mostra a cena simulando a realidade, onde os objetos se tornam menores quanto mais distantes estiverem de um observador.

Quando um objeto na aba *Scene* estiver selecionado, é possível fazer algumas alterações neste, através das teclas W para posição, E para rotação e R para escala.

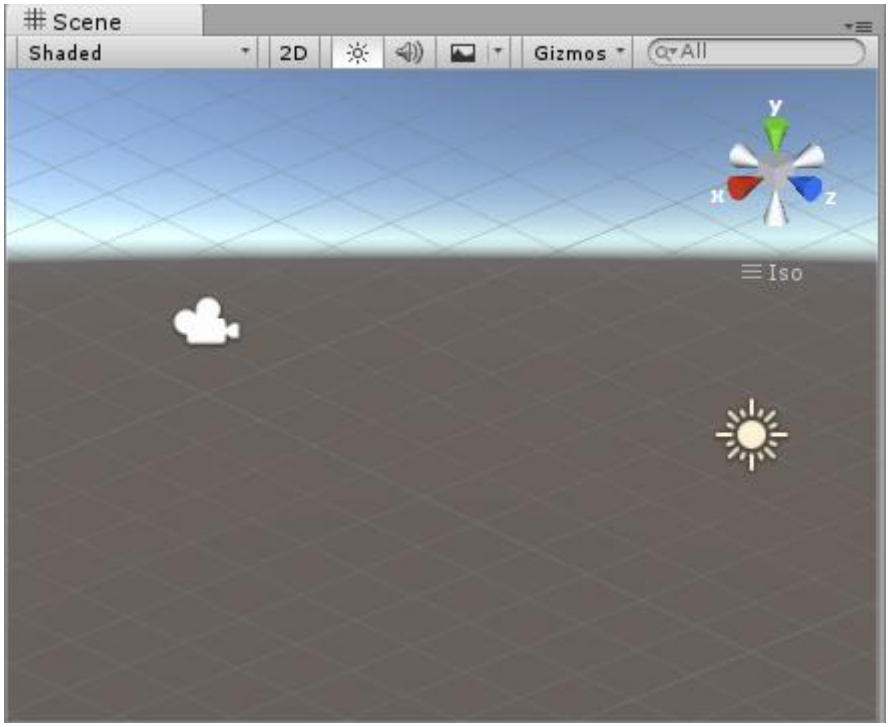


Figura 2.2. Aba *Scene*

2.2.1.3 Aba *Game*

Esta é a aba responsável por mostrar ao desenvolvedor como **será** a visão do jogo, quando este estiver sendo executado fora da Unity. Ou seja, é a maneira como o jogo será visto pelas pessoas que forem jogar.

Basicamente, o que o jogador consegue enxergar é tudo o que estiver no campo de visão de pelo menos uma câmera, e que possua alguma componente de renderização. Por exemplo, se temos um objeto que possui **apenas** uma componente de colisão e este objeto estiver posicionado na frente de alguma câmera, ele não será renderizado na aba *Game*.

Os componentes mais básicos que habilitam a renderização de um objeto são o ***Mesh Renderer*** e o ***Sprite Renderer***. O primeiro serve para renderizar malhas 3D, como por exemplo cones, cubos, ou até mesmo um dragão, todos em 3D. O segundo serve para renderizar *sprites*, que basicamente são imagens 2D. Existem também outros tipos de componentes que possibilitam a renderização, mas que não serão objetivo deste texto.

Portanto, quando qualquer objeto que possuir alguma das componentes citadas acima, caso estas componentes estejam preenchidas com a malha ou imagem a ser renderizada e estiver na frente de alguma câmera, este objeto aparecerá na aba *Game*.

Seguimos com a explicação de alguns dos botões presentes nesta aba. Podemos verificar os botões na Figura 2.3.

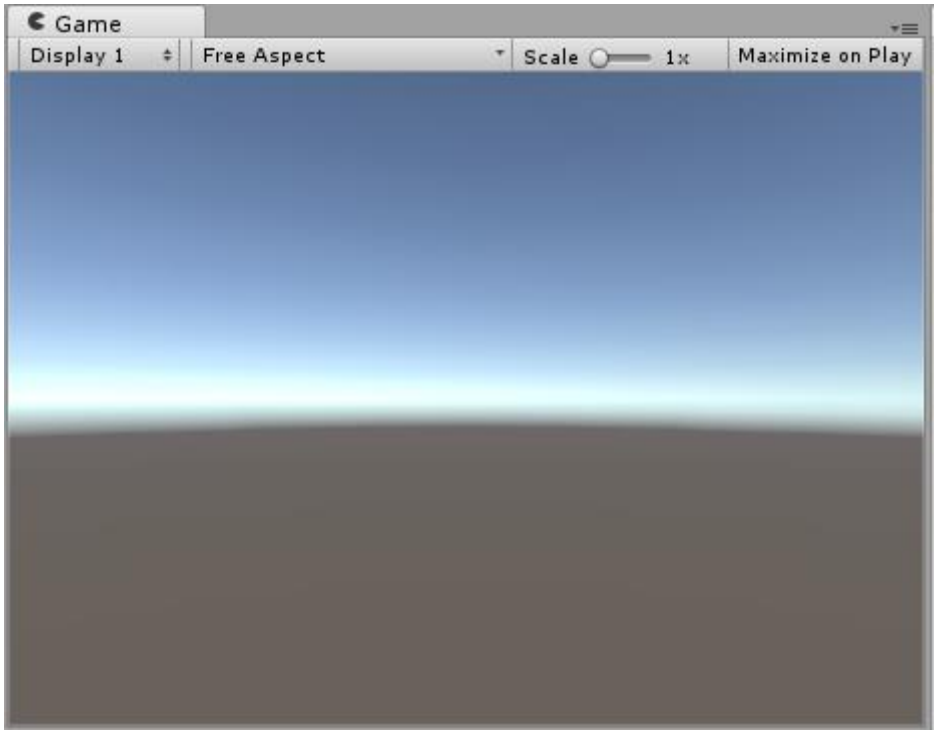


Figura 2.3. Aba *Game*

Clicando em "*Free aspect*", abrimos um sub-menu com os possíveis *Aspect Ratios* que mede o formato da tela do jogo. Dois bons exemplos de *aspect ratio* são o 4:3, que é um formato mais quadrado tal como os das televisões antigas, e o *aspect* 16:9, um formato mais largo na horizontal, que chamamos de *widescreen*. Dependendo do tamanho da tela do equipamento no qual o jogo irá ser executado, o desenvolvedor deve escolher um aspecto que melhor se encaixar.

Ativando o botão "*Maximize on Play*" faz com que, sempre que o desenvolvedor clicar no botão de *play*, a aba *Game* seja maximizada.

É importante também perceber que tudo o que aparece nesta aba são os objetos que estiverem sendo renderizados por pelo menos uma câmera, como dito anteriormente. Portanto, caso não exista nenhuma câmera no jogo ou, por algum motivo, todas as câmeras foram desativadas, a imagem ficará toda escura com uma mensagem de erro sendo exibida, como pode ser observado na Figura 2.4.

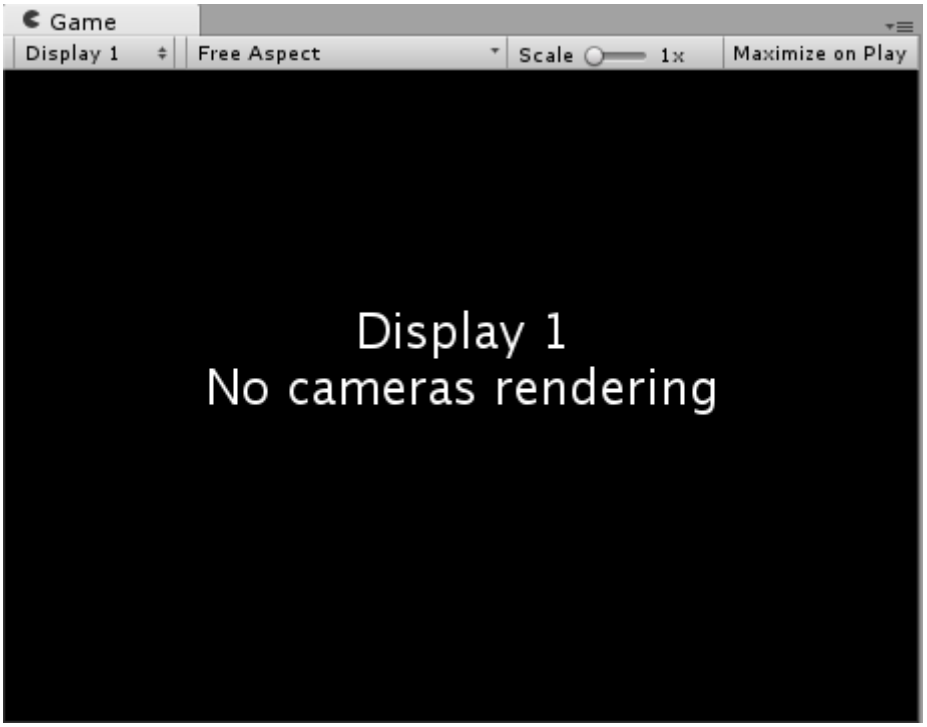


Figura 2.4. Aba *Game* sem câmeras ativas

2.2.1.4 Aba *Hierarchy*

É a aba responsável por informar quais objetos existem na cena e como estes estão organizados entre si. Isso significa que, por mais que tenhamos o *asset* de um robô na nossa pasta *Assets*, este robô não existe ainda dentro do nosso jogo enquanto não estiver na hierarquia.

Através desta aba, é possível criar *assets* da mesma maneira que criamos na aba *Project*, porém estes já serão adicionados diretamente na hierarquia, portanto já existirão diretamente na cena do jogo.

Além de informar os objetos existentes na cena, é possível relacionar os objetos entre si, criando uma estrutura de "objeto pai" e "objeto filho". Com isso, podemos criar grupos de objetos e efetuar operações diretamente em grupos. Por exemplo, todas as transformações (mudança de posição, de rotação e de escala) que forem aplicadas em um objeto pai serão também aplicadas a qualquer objeto filho deste. Assim, podemos mover grupos de objetos simultaneamente, se necessário.

Um exemplo de estrutura pai-filho é exibido na Figura 2.5, onde o objeto *Scenario* é pai dos objetos *House01*, *House02* e *Ground*.

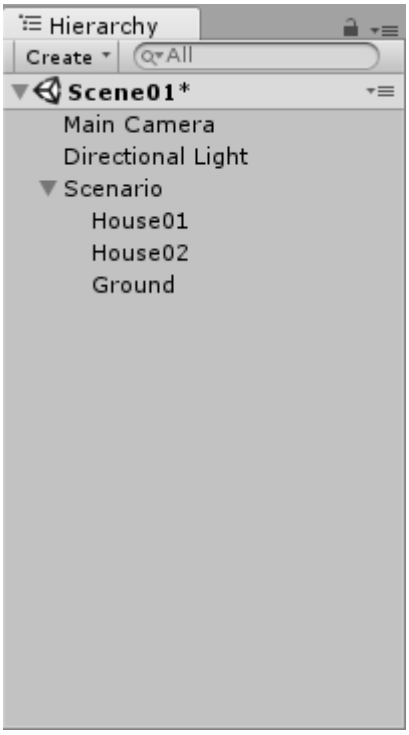


Figura 2.5. Aba *Hierarchy*

2.2.1.5 Aba *Inpector*

Basicamente, serve para configurar objetos e *assets*. Para objetos, o inspetor irá iniciar com o nome do objeto, um *check-box* intitulado "*static*", que serve informa se o objeto será ou não estático. Temos também um menu *Tag* e o *Layer*. Podemos separar os objetos através de diferentes *tags* e realizar diferentes ações para objetos que possuem diferentes tags e, através de códigos, podemos tomar diferentes ações para objetos que possuem determinada tag.

Podemos também separar os objetos em diferentes *layers* e aplicar diversas regras nestas *layers*, como regras de colisão e de renderização.

Abaixo deste primeiro painel que analisamos e que podemos observar na Figura 2.6, a aba *inspector* irá listar todos os componentes que aquele objeto possuir. Componentes são tudo o que definem um objeto. Por exemplo, se temos um objeto que possui uma imagem 2D e um colisor, os componentes para este objeto serão: ***Transform***, ***SpriteRenderer*** e ***Collider2D***. O primeiro irá informar qual a posição, rotação e escala do objeto no mundo 3D, o segundo irá informar qual é a imagem 2D que será renderizada naquela posição e o terceiro informa qual é o colisor que aquele objeto tem. Pode-se observar estes componentes na Figura 2.6.

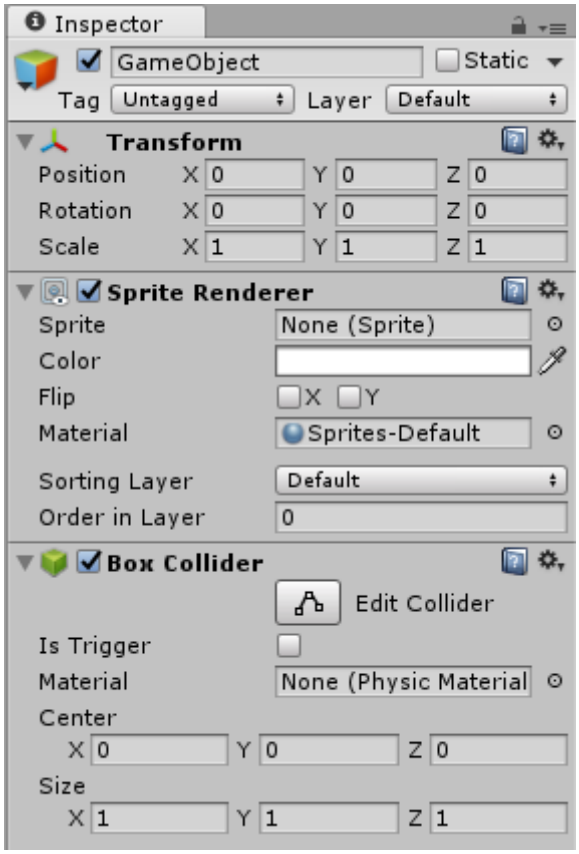


Figura 2.6. Aba *Inspector*, exemplos de componentes

Outra possibilidade na aba *Inspector* é a de adicionar componentes, como demonstrado na Figura 2.7.

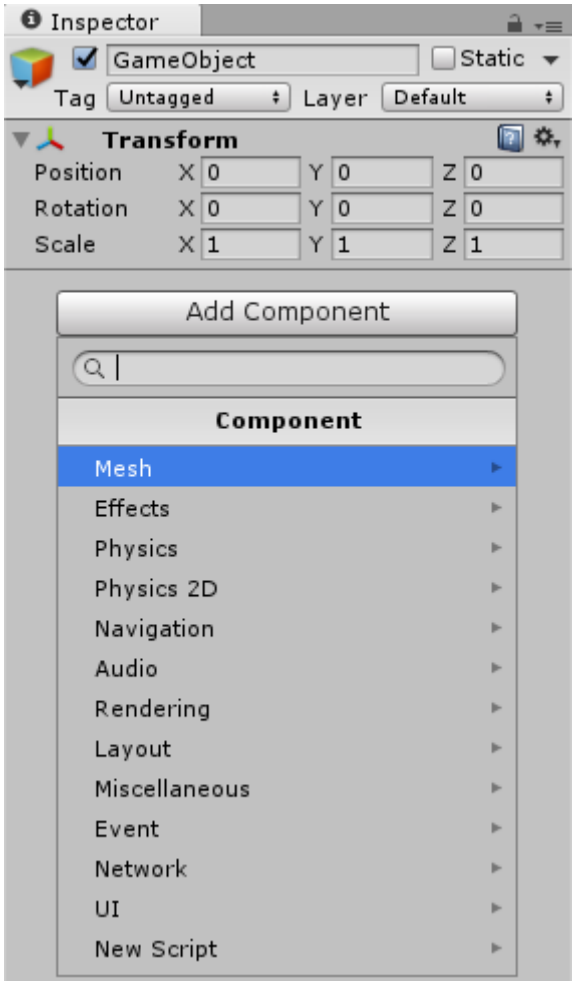


Figura 2.7. Aba *Inspector*, *Add Component*

2.2.2. *Scripting*

Na Unity3D, podemos programar em duas linguagens de programação: C# e UnityScript (comumente chamado de JavaScript).

Decidir qual linguagem de programação é melhor para utilizar depende inteiramente da experiência e das vontades do programador. Se o programador já tem costume com C++ ou Java, então programar em C# trará diversas semelhanças. Se o programador já tem experiência com JavaScript ou PHP, por exemplo, então este provavelmente terá mais facilidade com UnityScript.

2.3. Implementando o jogo

Iremos iniciar a implementação do nosso jogo através da preparação dos *assets* que iremos utilizar. Como mencionado anteriormente, o objetivo deste texto não é de criar modelos 3D, texturização, áudio, etc. Portanto, iremos buscar *assets* de outros autores. Para isto, devemos falar sobre a loja virtual da Unity.

2.3.1. Unity *Asset Store*

A *asset store* um espaço exclusivo da Unity para a distribuição de *assets* online, sejam gratuitos ou não. Lá nós encontramos todo o tipo de *asset*, como modelos 3D, animações, sistemas de partículas, texturas e muito mais.

Neste minicurso utilizaremos pacotes que contém: um monstro, um conjunto de casas, objetos medievais, uma arma, um efeito sonoro de disparo e alguns scripts, tudo podendo ser encontrado diretamente na *Asset Store*. Porém, não é estritamente obrigatório o uso dos mesmos materiais aqui apresentados. Você pode desenvolver o seu próprio material, ou procurar outros na loja virtual da Unity.

2.3.2. Criando um novo projeto

Para iniciar a implementação do nosso jogo, o primeiro passo é abrir a Unity3D. A versão desta que será utilizada neste texto é a versão **5.4.0f3**. Na tela inicial, clique em ***New***.

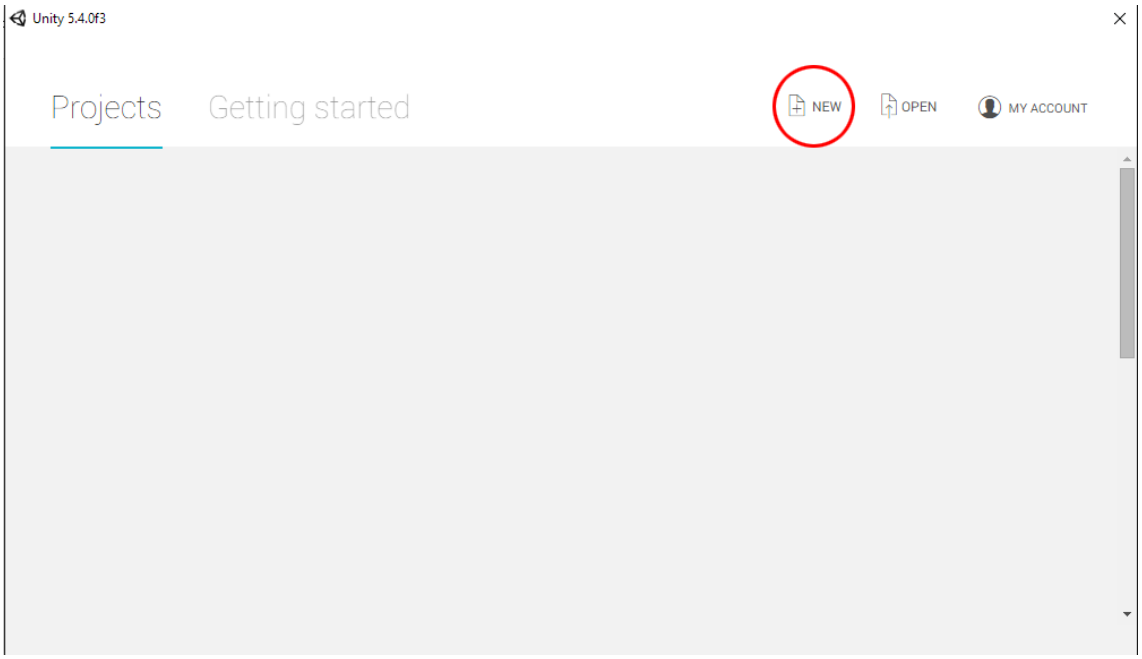


Figura 2.8. Unity3D, tela inicial

A seguir, informe qual será o nome do seu projeto, onde este será localizado no seu computador, se o seu projeto será 3D ou 2D e se você deseja adicionar algum pacote de *assets* ao projeto, durante o ato de criação. Se você já possuir os *assets* no seu computador, você já pode adicioná-los. Caso contrário, a adição de pacotes poderá ser feita posteriormente na aba *Project*, como informado anteriormente.

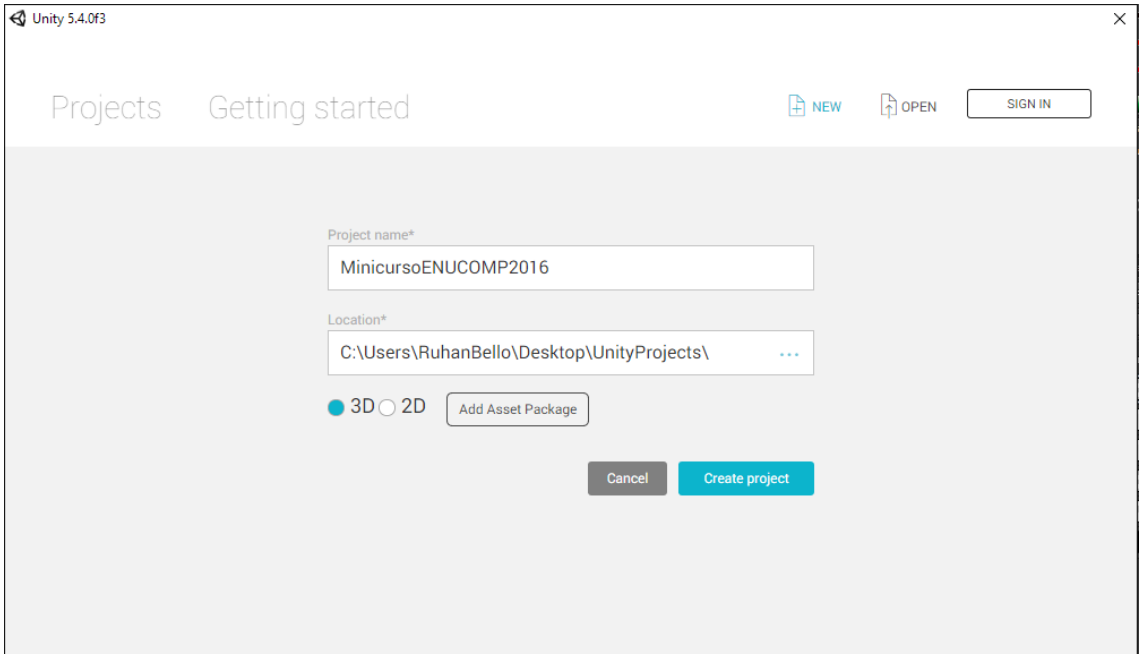


Figura 2.9. Unity3D, *New Project*

Clique no botão ***Create Project*** e espere enquanto a Unity cria o seu projeto. Será criada uma pasta na localização informada em ***Location*** e o nome da pasta será o que foi informado em ***Project Name***.

2.3.3. Criando o cenário

Por motivo de segurança, assim que o projeto for aberto na Unity, clique com o botão direito na pasta *Assets* da aba *Project* e vá em ***Create -> New Folder***. Dê o nome *Scenes* ao novo diretório. A seguir, no painel superior vá em ***File -> Save Scene*** (ou **ctrl + S**) para salvar a sua cena, mesmo que ainda esteja vazia. Salve na pasta *Scenes* que você acabou de criar dê o nome *MainScene* à cena.

Vale ressaltar que é possível a criação de mais de uma cena para um jogo, como uma cena de *loading*, uma cena de *gameplay*, uma de *game over*, etc.

Neste momento do projeto, você já deve ter os seus *assets* (monstros, cenários, etc) dentro da pasta *Assets* também. Caso não tenha, essa é a hora de importa-los clicando com o botão direito na pasta e indo em *Import New Asset* ou *Import Package* ou através da *Asset Store*.

Agora, vamos posicionar os elementos na nossa cena. Na aba *Hierarchy*, clique com o botão direito e em ***Create Empty*** para criar um novo objeto vazio. Dê o nome *Scenario* para este objeto, que será utilizado apenas para organizar o cenário que iremos montar, colocando todos os objetos do cenário como filhos do objeto *Scenario*. A seguir, na aba *Inspector*, certifique-se de que os valores em *Position* estão todos iguais a zero, como demonstrado na Figura 2.10.

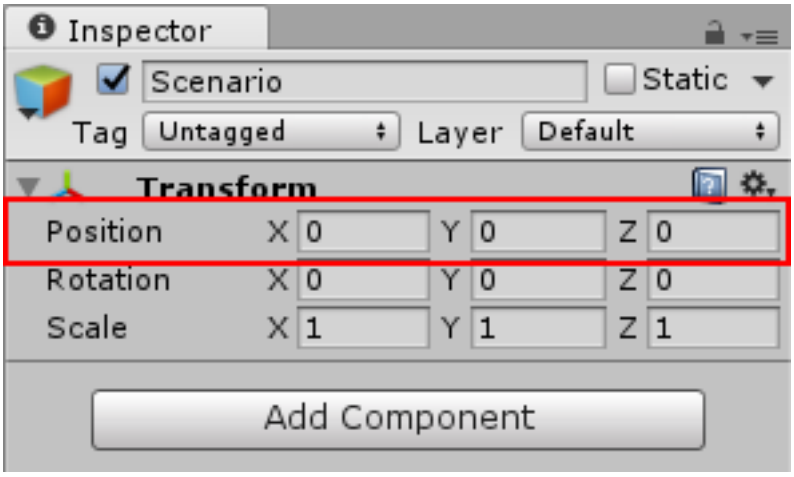


Figura 2.10. *Scenario* com posição zerada

Em seguida, vá na pasta *MedievalTownExteriors* > *Terrain*, clique e arraste o *asset Demo_Terrain* para algum espaço vazio da aba *Hierarchy*. Com isso, acabamos de criar uma instância do *asset* na nossa cena do jogo. Clique e arraste esta instância para dentro do objeto *Scenario*, para que o terreno seja filho deste. A seguir, posicione o terreno no ponto: X = -250, Y = 0 e Z = -250.

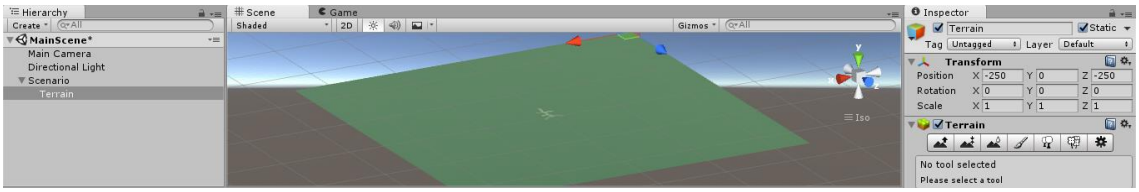


Figura 2.11. *Terrain* com posição configurada

Na pasta *MedievalTownExteriors* > *Prefabs*, encontramos diversos objetos que podemos utilizar para compor nossa cena. Aqui, fique à vontade para posicionar estes na sua cena como quiser. Neste texto, colocaremos um barril, uma casa e três pedras. Para instanciar um destes objetos na cena, clique e arraste-o até a aba *Hierarchy*, para **dentro** do objeto *Scenario*. Para posicionar, defina uma posição na aba *Inspector*, ou utilize as setas que surgem quando você seleciona um objeto, para movê-los nos eixos X, Y e Z. Posicione-os próximos à marca central do nosso terreno. Certifique-se de que a posição do seu objeto é **sempre zero no eixo Y**, assim como a do terreno, para que os objetos fiquem em contato com o chão e não flutuando.

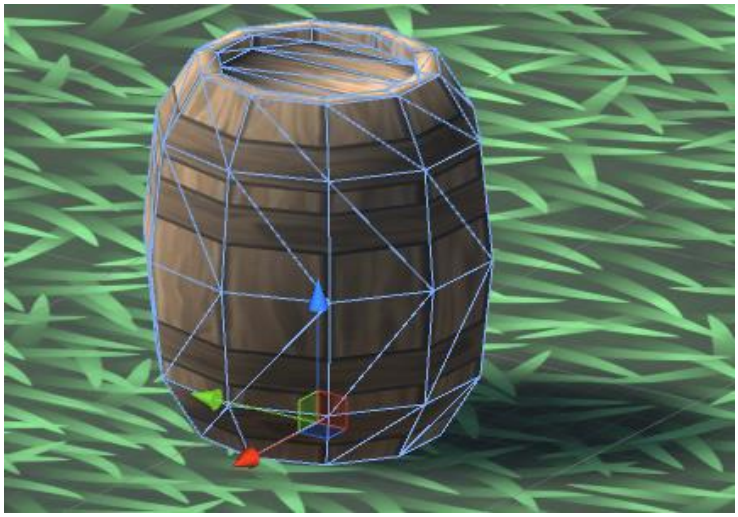


Figura 2.12. Posicionando objetos

Segue o resultado do posicionamento dos objetos que foram citados acima.

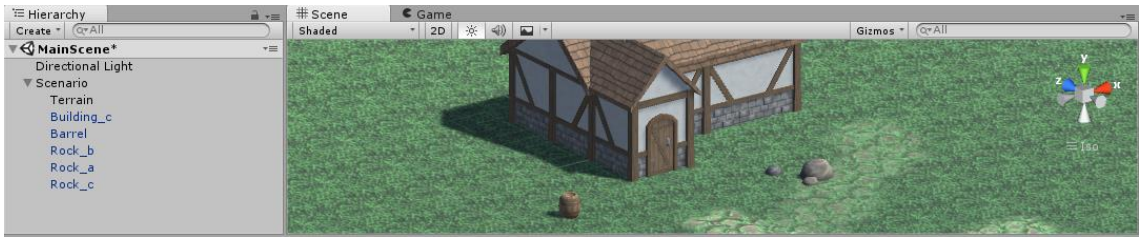


Figura 2.13. Cenário finalizado

2.3.4. Criando o jogador

A princípio, na aba *Hierarchy*, clique no objeto *Main Camera* e delete-o. Não iremos precisar da câmera padrão criada pela Unity pois usaremos outra câmera, que vem junto de um dos *assets*.

É importante que, neste projeto, só exista uma câmera no jogo, pois se houver mais de uma, a visão das duas câmeras será mesclada em uma só e a imagem resultante disto é o que será mostrado ao jogador, o que pode causar efeitos indesejáveis, por este motivo deletamos a câmera padrão criada pela Unity.

Vá na pasta *Standard Assets* > *Character Controlles* e arraste o *First Person Controller* para algum espaço vazio da aba *Hierarchy*. A seguir, posicione este novo objeto no ponto X = 0, Y = 1 e Z = 0. Explorando este objeto que acabamos de criar, vemos que ele possui dois objetos filho: *Graphics* e *Main Camera*.

Graphics é o objeto responsável por informar qual objeto vai ser renderizado na posição onde o nosso jogador estiver, por isso ele contém a componenteb *Mesh Renderer* e o objeto a ser renderizado é uma espécie de cápsula.

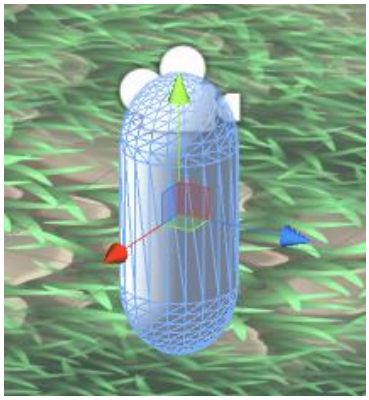


Figura 2.14. *First Person Controller*, *Graphics*

MainCamera é semelhante à câmera padrão criada pela Unity, mas que já vem com algumas alterações necessárias para o controle do nosso personagem, como a componente *Mouse Look*.

O objeto *First Person Controller* é composto também por várias componentes e será um dos responsáveis pelo controle do nosso personagem, tal como a colisão deste com outros objetos. Observe que este objeto possui a componente *CharacterController*, que é uma componente que define um colisor para o nosso personagem, e graças a este colisor ele é capaz de pisar no chão, esbarras nas pedras, na casa, etc. O sistema de colisão, resumindo, é um sistema que está sempre observando todos os objetos que possuem colisores na cena. Sempre que o colisor de um objeto encostar no colisor de outro objeto, um evento de colisão é disparado. Neste momento, os objetos se "esbarram" e não podem atravessar um ao outro. Porém, é possível que dois objetos que possuem colisor consigam atravessar um ao outro e isto vai ser útil para nós, mas voltaremos a esta questão posteriormente.

Pressionando no botão de *Play*, podemos executar o nosso jogo e ver como ele está ficando e como está a visão do jogador, na aba *Game*. Até o momento, nós já temos um personagem controlável através do mouse para rotacionar a visão, das teclas WASD e das setas direcionais para movimentação, e da barra de espaço para saltar. Veja que já é possível que o nosso jogador esbarre nos objetos.

2.3.5. *Shooting*

Já temos um personagem controlável, mas este ainda não atira. Sistemas de tiro funcionam utilizando colisões. Primeiramente, precisamos que todo objeto que seja "atirável" tenha um colisor em sua volta, pois quando o jogador der tiros, precisamos saber se a bala colidiu com algo e, se colidiu, identificar com quem a bala colidiu e a partir desta informação tomar alguma decisão.

Neste texto, a mecânica de tiros que será implementada é a que chamamos de ***Raycasting***. *Raycasts* são, basicamente, "raios" ou linhas finíssimas que tem um ponto de origem e estendem-se até um ponto final. Isto se torna útil no nosso jogo de tiro porque cada tiro que houver será um *raycast* que sai da ponta da arma do nosso personagem e percorre certa distância adiante, ou seja, é uma maneira interessante de implementar um tiro pois balas (geralmente) seguem em linha reta a uma velocidade absurda até encontrar um alvo.

Existe outra característica importantíssima dos *raycats* que os torna tão útil para o nosso jogo: a característica de detectar colisão. Sempre que um *raycast* atinge qualquer colisor, ele captura as informações sobre aquela colisão naquele momento. Então, a partir destas informações, é que tomamos decisões, dependendo do que o nosso *raycast* tiver atingido.

Para implementar este sistema, primeiro vá até a pasta *WeaponsHQ4 > Prefabs* e arraste o *asset Pistol01* até o objeto *MainCamera*, contido no objeto *First Person Controller*. Na aba *Hierarchy*, selecione o objeto *Pistol01* e altere sua posição para: X = 0.25, Y = -0.1 e Z = 0.5. Altere sua rotação para X = 0, Y = 90 e Z = 0, para que a arma esteja apontando para frente, e esteja posicionada na frente do nosso personagem.

A seguir, crie no seu projeto uma pasta chamada *Scripts*. Dentro desta pasta, clique com o botão direito do mouse e vá em *Create > C# Script*. Nomeie como *RaycastShoot*. A seguir, na aba *Hierarchy*, clique no objeto *Pistol01* e na aba *Inspector* clique em *Add Component*. Digite *RaycastShoot* e quando aparecer o script C# que acabamos de criar, selecione-o. Assim a arma do personagem possui o nosso novo script como uma componente.

Antes de entrarmos em detalhes sobre linhas de código, vamos adicionar nos nossos objetos as componentes necessárias.

Vá no editor e clique no objeto *Pistol01*. Em seguida, clique em *Add Component* para adicionar as componentes *Line Renderer* e *Audio Source*. Ao adicionar a componente *Line Renderer*, procure nesta o campo *Parameters* e modifique os valores de *Start Width* e *End Width* para 0.05. **Desabilite** a componente logo em seguida, clicando na *checkbox* ao lado do seu nome, deixando o seu valor como falso.

Quanto ao *Audio Source*, clique no pequeno círculo ao lado do campo *AudioClip* e selecione o arquivo de áudio *C02 Shooter B Gun Shot Fire 2 - Nova Sound*.

Em seguida, clique com o botão direito no objeto *Pistol01*, na aba *Hierarchy* e em seguida clique em *Create Empty*. Selecione o novo objeto criado e coloque o nome *GunEnd* neste. Posicione *GunEnd* no ponto X = -0.13, Y = 0.03 e Z = 0.

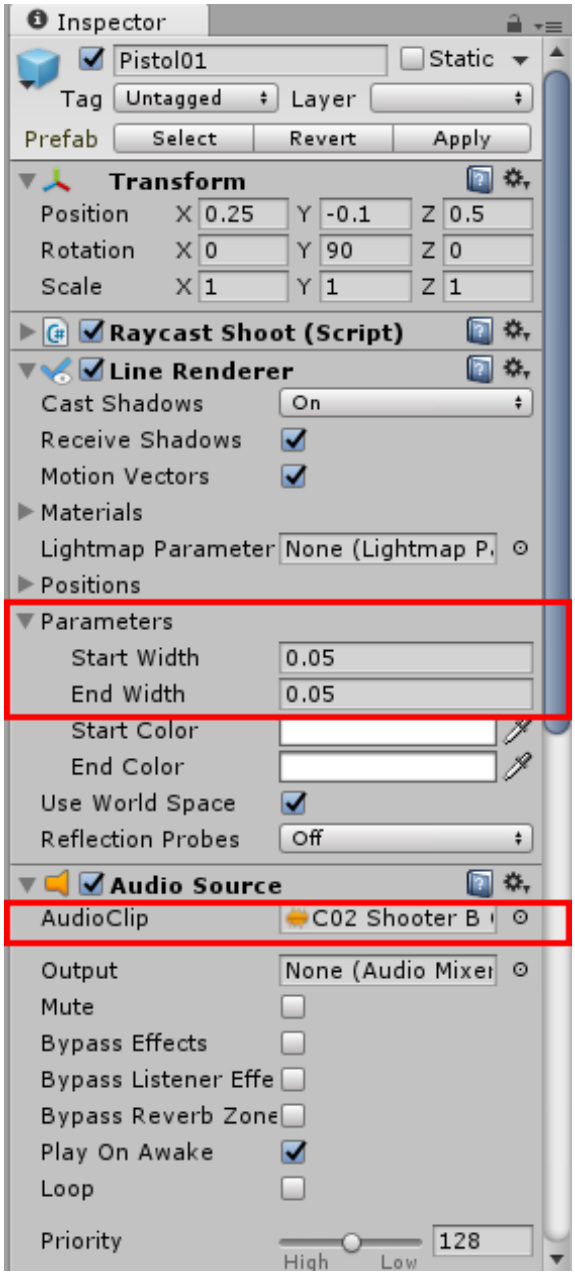


Figura 2.15. Componentes *Line Renderer* e *Audio Source*

Por fim, vamos adicionar ao nosso jogo uma mira, para que o jogador tenha uma melhor noção do ponto que será atingido quando este realizar um disparo. Para isso, vá no editor, clique com o botão direito em algum espaço **vazio** da hierarquia e clique em *UI* > *Canvas*. Em seguida, clique com o botão direito no objeto que foi criado e clique em *UI* > *Image*. Com o objeto *Image* selecionado, vá na aba *Inspector* e na componente *Rect Transform* defina a posição X = 0, Y = 0 e Z = 0. Defina *Width* = 5 e *Height* = 5. Pronto! Agora, para onde quer que o jogador mova a câmera, sempre haverá um pequeno ponto branco no centro da tela, indicando a direção dos disparos.

Parte dos códigos que serão utilizadas neste texto vêm diretamente de tutoriais da própria Unity3D. Analisaremos o código exposto nas Figura 2.16 em diante, entendendo a funcionalidade de cada linha do código.

```
4 public class RaycastShoot : MonoBehaviour {
```

Figura 2.16. Classe *RaycastShoot*, declaração

Na Unity, sempre que criamos um novo script em C#, uma nova classe é criada estendendo de *MonoBehaviour*, uma classe base da qual todo script deriva.

```
6     public int gunDamage = 1;
7     public float fireRate = .25f;
8     public float weaponRange = 50f;
9     public Transform gunEnd;
```

Figura 2.17. Classe *RaycastShoot*, Variáveis públicas

A seguir, temos este conjunto de variáveis públicas. Antes de explicar cada uma delas, deve-se atentar a uma das principais diferenças entre variáveis públicas e privadas quando utilizadas na Unity, além das diferenças comuns implicadas pela própria linguagem.

Variáveis públicas na Unity possibilitam que o valor daquela variável seja alterado através do *Unity Editor*.

Figura 2.18. Variáveis públicas no *Unity Editor*

Observe que todas as variáveis públicas demonstradas na Figura 2.17 mostram-se alteráveis no *Unity Editor* como demonstrado na Figura 2.18. Ou seja, os valores das variáveis podem ser alterados desta maneira, sem precisar alterar linhas de código.

A variável *GunDamage* servirá para informar qual é o dano que um tiro da pistola causa em um monstro.

FireRate informa com que frequência o jogador poderá atirar, para limitar a velocidade dos tiros que o jogador pode dar.

WeaponRange é o limite que uma bala da pistola pode alcançar.

GunEnd será uma referência ao local onde fica a ponta da arma para que, desta maneira, quando dispararmos com a pistola, o efeito da bala saia exatamente da ponta da pistola. Se você verificar no *Unity Editor*, todas as variáveis públicas estão lá inicializadas, exceto a variável *GunEnd*. Para inicializa-la, clique no círculo ao lado direito do nome da variável e selecione o objeto *GunEnd*.

```
12      private Camera fpsCam;
13      private WaitForSeconds shotDuration = new WaitForSeconds(.07f);
14      private AudioSource gunAudio;
15      private LineRenderer laserLine;
16      private float nextFire;
```

Figura 2.19. Classe *RaycastShoot*, Variáveis privadas

Variáveis privadas por sua vez não aparecem no *Unity Editor*, portanto só podem ser alteradas via código, por isso elas não aparecem na Figura 2.18.

A variável *fpsCam*, que é do tipo *Camera* serve para guardar uma referência à componente *Camera*, localizada no objeto *MainCamera* que está acoplada ao personagem, para que dessa maneira algumas informações desta possam ser utilizadas.

ShotDuration determina o tempo que vai durar o efeito do nosso tiro, pois para mostrarmos visualmente a direção para onde a nossa bala está indo, iremos utilizar o desenho de uma linha. *ShortDuration* informação por quanto tempo este efeito estará ativo. Variáveis do tipo *WaitForSeconds* são utilizadas em funções especiais cujo retorno é do tipo *IEnumerator*. Exploraremos este tipo de função mais adiante.

GunAudio guardará uma referência a uma componente *AudioSource* acoplada à nossa pistola. *AudioSources* são fontes de áudio, ou seja, basicamente são componentes que fazem com que um dado objeto possa emitir sons. Esta variável será utilizada para fazer com que o *AudioSource* da pistola emita o som de tiro.

LaserLine será utilizada para gerar o efeito visual de nossa bala.

NextFire informará quando que o usuário este poderá dar o próximo tiro. Esta variável será utilizada em conjunto com a variável pública *FireRate*.

Quando criamos códigos na Unity3D, devemos compreender algumas das funções que a própria *engine* possui como padrão. Algumas das mais importantes delas são: *Awake*( ), *Start*( ) e *Update*( ).

Estas funções são chamadas por padrão em uma ordem pré-definida. A função *Awake* é chamada quando a instância daquele script é carregada. A função *Start* é chamada antes do primeiro *frame* ser processado. A função *Update* é chamada uma vez a cada *frame*. Considerando esta ordem, é comum que inicializações de variáveis sejam

feitas na *Awake* e na *Start*, por acontecerem exatamente no começo do jogo quando este é executado. É natural utilizar a *Update* para modificar ou ler informações a cada *frame*.

```
18      // Use this for initialization
19      void Start () {
20
21          laserLine = GetComponent<LineRenderer> ();
22          gunAudio = GetComponent<AudioSource> ();
23          fpsCam = GetComponentInParent<Camera> ();
24
25      }
```

Figura 2.20. Classe *RaycastShoot*, *Start*

A função *Start* neste caso está sendo utilizada para iniciar três das nossas variáveis, que até o momento permaneciam com valor nulo. Deve-se notar que as únicas variáveis que foram inicializadas no método *Start* são aquelas cujos tipos não são nativos da linguagem C#, ou seja, apenas as variáveis de tipos específicos da Unity.

LineRenderer, *AudioSource* e *Camera* são componentes que foram ser adicionadas a objetos. Portanto, se temos variáveis destes tipos e queremos que estas variáveis referenciem as componentes do próprio objeto, precisamos executar a linha de código *GetComponent* para que a referência seja feita. Quando a linha 21 é executada, o nosso script informa ao sistema que procure naquele object por alguma componente do tipo *LineRenderer* e, se encontrar, a variável *laserLine* guarda a referência a esta componente. Dessa maneira, se o nosso objeto possuir uma componente do tipo *LineRenderer*, agora nós podemos alterar os valores da própria componente através do nosso script, sem necessitar mexer em algo no *Unity Editor*. Vale lembrar que o script que estamos analisando está no objeto *Pistol01*, que é **objeto filho** de *MainCamera*. Por isso adicionamos, anteriormente, a componente *LineRenderer* a ele.

Da mesma maneira é obtida a referência a ser guardada na variável *GunAudio*, na linha 22. Na linha 23, entretanto, temos o método *GetComponentInParent*, que funciona de maneira similar ao *GetComponent*, mas que procura a componente no objeto pai ao invés de no próprio objeto e, como sabemos, a nossa *MainCamera* é pai do objeto *Pistol01*.

Na Unity, uma das maneiras que utilizamos para verificar se o usuário pressionou algum botão é perguntando, a cada *frame*, se um determinado botão foi pressionado. Considerando que é um tipo de verificação que nós fazemos a cada *frame*, fazemos isto dentro da *Update*.

```
27      // Update is called once per frame
28      void Update () {
29
30          if (Input.GetButtonDown ("Fire1") && Time.time > nextFire) {
31
32              nextFire = Time.time + fireRate;
33
```

Figura 2.21. Classe *RaycastShoot*, *Update* e *Input*

Na linha 30, onde temos *Input.GetButtonDown*, estamos verificando se, naquele frame, um determinado botão acabou de ser pressionado pelo jogador. O botão que procuramos neste caso é o que está identificado como *Fire1*. Para sabre qual botão é identificado por *Fire1*, basta ir no *Unity Editor* e no painel superior clicar em *Edit > Project Settings > Input*. Isto irá abrir, na aba *Inspector*, um painel chamado *Axes*, que é onde os botões são identificados através de nomes. Observando este painel, observamos a existência de *Fire1*, que é justamente o que estamos procurando através da linha 30 do código. Expandindo *Fire1*, existem dois botões que respondem por este nome: o *left ctrl* e o *mouse 0*. *Mouse 0* representa o botão esquerdo do mouse. Ou seja, o que estamos observando na linha 30 é: "O jogador pressionou o botão *left ctrl* do teclado, ou o botão esquerdo do *mouse*?". Se esta condição for verdadeira, as instruções da condição serão executadas.

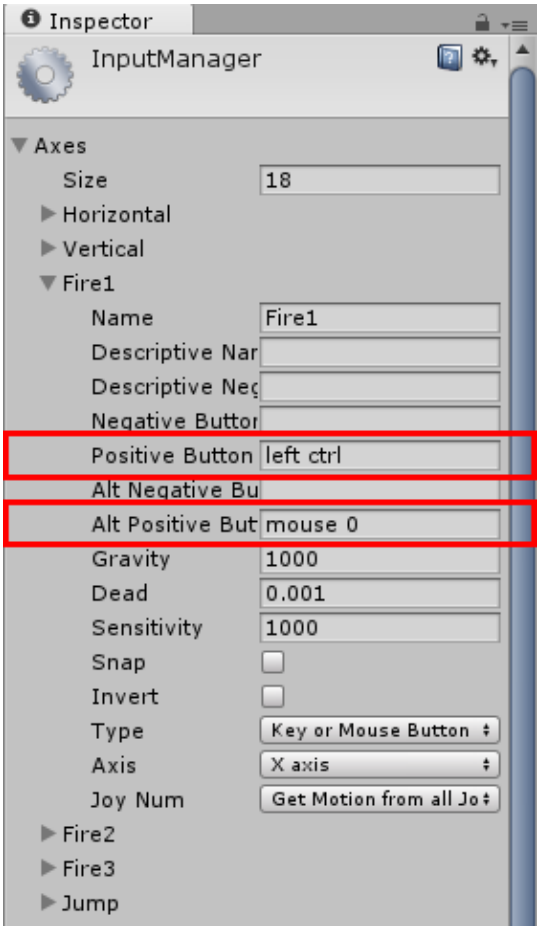


Figura 2.22. *Unity Editor*, *Axes*

Ainda na linha 30, temos mais uma condição que iremos explorar logo adiante. Antes, devemos nos atentar à linha 32. Vale lembrar que a variável *NextFire* serve para guardar o momento em que o jogador poderá efetuar o seu próximo tiro. Ou seja, quando fazemos "*NextFire = Time.time + FireRate*", significa que o momento em que o jogador poderá efetuar o seu próximo tiro é: o tempo atual do jogo (*Time.time*) somado com a taxa de tiros (*FireRate*). Considerando que *FireRate* foi inicializado como 0.25, o que esta linha de código diz é: "o jogador só poderá atirar novamente quando se passarem 0.25 segundos do momento atual do jogo.

Isto se completa com aquela outra condição da linha 30: *Time.time* > *nextFire*. Esta linha verifica se o tempo atual do jogo já superou o tempo determinado pela variável *NextFire*, ou seja, se já se passaram os 0.25 segundos. Deve-se notar que a variável *NextFire* é atualizada sempre que o jogador consegue dar um tiro com sucesso, ou seja, quando as duas condições da linha 30 retornarem *true*.

```
34            StartCoroutine (ShotEffect());
```

Figura 2.23. Classe *RaycastShoot, StartCoroutine*

Na Unity, nós temos funções cujo retorno é do tipo *IEnumerator*. Este tipo de função serve, basicamente, para contar tempo. Nota-se que através das variáveis *NextFire* e *Time.time* nós fomos capazes de contar tempo, mas utilizar funções do tipo *IEnumerator* é uma outra maneira de fazer isto. Esse tipo de função também é conhecido como corrotina, ou c*oroutine* e suporta **mais de um** retorno do tipo ***WaitForSeconds***. No corpo deste tipo de função, o seguinte comando: *yield return new WaitForSeconds*(5) faz com que 5 segundos sejam esperados antes de seguir pra próxima linha de código. Ou seja, o parâmetro informado em *WaitForSeconds*() determina quanto tempo deve ser esperado antes de prosseguir.

Para iniciar uma corrotina, é necessário chama-la através do método *StartCoroutine(NomeDaCorrotina( ))*. No caso da linha 34, a corrotina que é iniciada se chama *ShotEffect()*, que analisaremos na figura a seguir.

```
55    private IEnumerator ShotEffect(){
56
57        gunAudio.Play ();
58
59        laserLine.enabled = true;
60        yield return shotDuration;
61        laserLine.enabled = false;
62
63    }
64
```

Figura 2.24. Classe *RaycastShoot, Coroutine*

Nessa corrotina, utilizamos a referência à nossa *AudioSource* para comandar que um áudio seja tocado. Em seguida, utilizamos a referência ao nosso *LineRenderer* para ativar o efeito de tiro. Em seguida, ordenamos que sejam esperados *ShotDuration* segundos, sendo que o valor contido nesta variável é *new WaitForSeconds*(.07f), ou seja, o tempo a ser esperado é de 0,07 segundos. Depois que esse tempo passa, desativamos o efeito de tiro.

```
36            Vector3 rayOrigin = fpsCam.ViewportToWorldPoint (new Vector3 (0.5f, 0.5f, 0));
37            RaycastHit hit;
38
39            laserLine.SetPosition (0, gunEnd.position);
```

Figura 2.25. Classe *RaycastShoot,* Origem do *raycast*

Como dito anteriormente, neste texto utilizaremos *raycast* para saber se uma bala atingiu um alvo. Ou seja, temos um "raio" que vai ter um ponto de origem e que vai se estender em uma direção até certa distância e verificar por colisões. Na linha 36, informamos qual é a origem do *raycast*. Utilizamos a referência que fizemos à *MainCamera* através da variável *fpsCam*, e buscamos, na visão da câmera, um ponto que esteja exatamente no seu centro. Dessa maneira, a origem do nosso *raycast* será sempre o ponto do meio da câmera, não importa para onde essa estiver apontado.

Em seguida, criamos a nossa variável *hit*, do tipo *RaycastHit*, que será utilizada posteriormente.

Na linha 39, começamos a criar o nosso efeito visual de disparo, que irá sair da ponta da nossa arma. Para isso, informamos que o início da linha que iremos renderizar é o valor que estiver contido na variável *GunEnd*, no atributo position.

```
41          if (Physics.Raycast (rayOrigin, fpsCam.transform.forward, out hit, weaponRange)) {
42
43              laserLine.SetPosition (1, hit.point);
44
45          } else {
46
47              laserLine.SetPosition (1, rayOrigin + (fpsCam.transform.forward * weaponRange));
48
49          }
```

Figura 2.26. Classe *RaycastShoot*, Disparando o *raycast*

Na linha 41, efetuamos de fato o *raycast*. Através do método *Physics.Raycast*, informamos, através dos parâmetros: a origem do raio, a direção que este irá seguir (no caso, *forward*, ou seja, "em frente"), em *out hit* nós recebemos o resultado do raio (ou seja, se este colidiu com algo), e em seguida informamos também o alcance do raio.

O método *Physics.Raycast* retorna verdadeiro caso o raio tenha colidido com algo. Ou seja, se ele colidiu com algo, o que nós queremos fazer é desenhar uma linha que inicia na ponta da nossa arma, até o lugar onde está o nosso alvo. O início da linha já foi determinado ante e agora é determinado o final da linha, que é informado em hit.point. Ou seja, o que informamos é: "busque o objeto com que o meu raio colidiu (que está contido em hit), e me informe o ponto onde este objeto se encontra". Este será o ponto final da linha.

Caso o método retorne negativo, nós queremos que o nosso efeito visual de disparo simplesmente siga em frente, de acordo com o alcance da nossa bala, o que é efetuado na linha 47.

2.3.6. Sistema de inimigos

Na aba *Project*, vá até a pasta *Zombie* > *Model*. Arraste o *asset z@walk* até a aba *Hierarchy* e posicione-o no ponto X = 0, Y = 0 e Z = 4. Coloque a rotação em X = 0, Y = 180 e Z = 0. Coloque a escala em X = 1.3, Y = 1.3 e Z = 1.3. Mude o nome do objeto para *Zombie*.

Em seguida, adicione uma componente do tipo *Box Collider* e preencha o campo *Center* com os valores X = 0, Y = 0.85 e Z = 0.2. Preencha o campo *size* com os valores

X = 0.5, Y = 1.6 e Z = 0.5. Este será o campo em que a nossa bala poderá atingir o zumbi, ou seja, é neste colisor que o *raycast* irá colidir.

Adicione uma componente do tipo *Rigidbody*. Todo objeto que possui esta componente terá a sua movimentação sujeita à física implementada pela Unity. Ou seja, a partir do momento em que você adiciona esta componente a um objeto, ele estará suscetível à força da gravidade, por exemplo.

Note que o zumbi já possui uma componente do tipo *Animator*. Esta componente é responsável por controlar animações em um objeto, ou seja, será necessária quando quisermos fazer o nosso zumbi andar e atacar. Componentes do tipo *Animator* precisam fazer referência a um *AnimatorController*, que é, basicamente, uma máquina de estados que determina em qual animação um objeto se encontra, e para quais animações ele poderá ir dependendo do seu controle.

Na pasta onde está o modelo do zumbi, clique com o botão direito e vá em *Create > Animator Controller*, coloque o nome *ZombieController*. Clique no zumbi e, na componente *Animator*, procure pelo campo *Controller*. Clique no pequeno círculo e selecione o *ZombieController* que acabou de criar. Observe as figuras a seguir para ver como deve ficar o objeto *Zombie*.

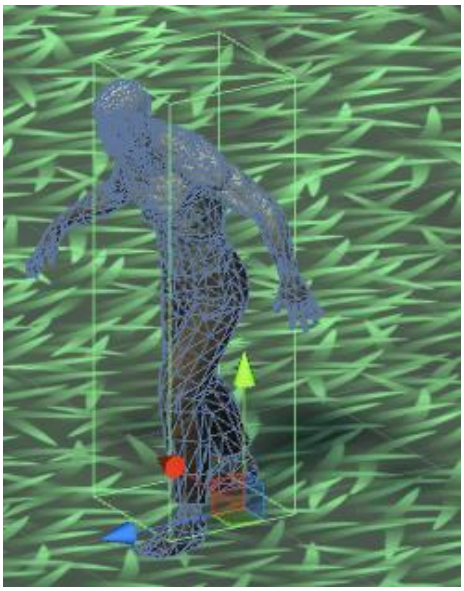


Figura 2.27. Zumbi na aba *Scene*

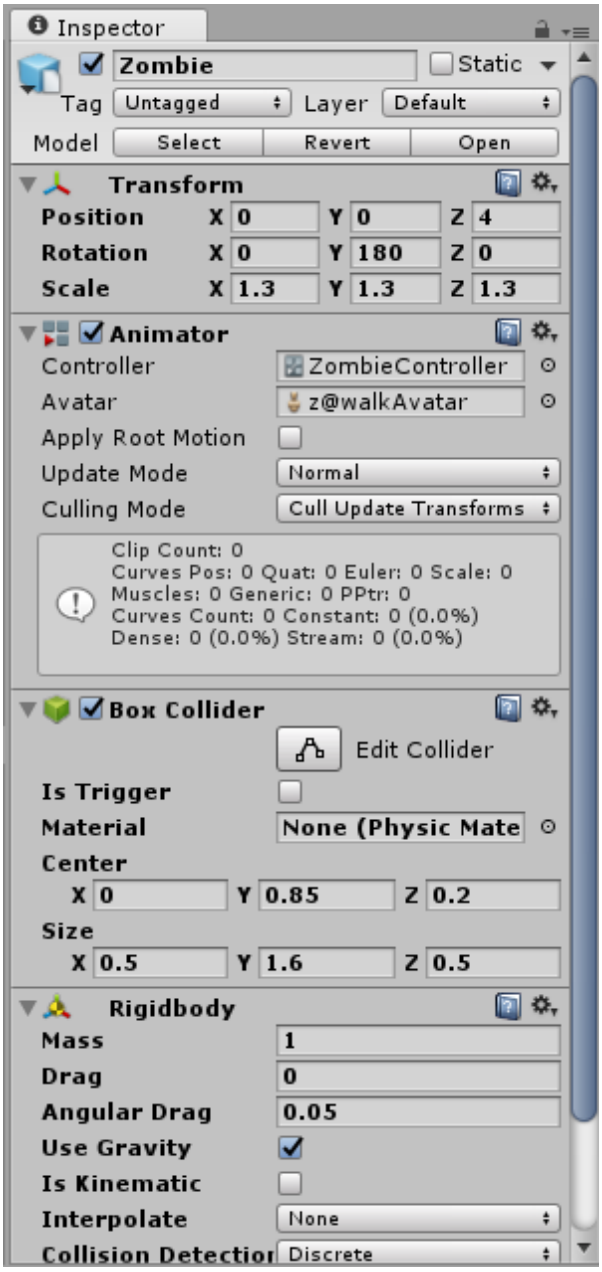


Figura 2.28. Configurações do zumbi

Em seguida, no painel superior, vá em *Window* > *Animation* e em seguida *Window* > *Animator*. Isto irá abrir duas novas abas no seu projeto. Ainda na pasta *Zombie* > *Model*, arraste os *assets z@walk* e o *z@attack* para dentro da aba *Animator*, nesta ordem. O que você vê nesta aba é a máquina de estados das animações. Existe uma seta que vai do estado *Entry* para o estado *walk*. Isso significa que a primeira animação a ser executada pelo zumbi será a animação de caminhar. Além disso, não existe nenhuma seta indo até o estado *attack*, significando que no momento não é possível chegar neste estado.

Para criar transições de um estado ao outro, clique com o botão direito do mouse em qualquer estado, depois em *Make Transation*. Isso fará surgir uma seta. Clique em

outro estado para informar a nova possível transição. Sabendo isto, possibilite a transição do estado *walk* para o estado *attack*, e a transição do estado *attack* para o estado *walk*. A seguir, ainda na aba *Animator*, clique em *Parameters*, no canto superior esquerdo da aba.

O menu *Parameters* serve para criar parâmetros que serão utilizados para efetuar as transições de um estado para o outro. Parâmetros podem ser do tipo *int*, *float*, *bool* e *trigger*. A transição que será possível para o nosso zumbi será a de "estar caminhando, e então atacar", quando ele se aproximar do nosso jogador. Para isso, crie um parâmetro do tipo *Bool* e nomeie-o *Attacking*.

Agora clique na seta que faz transição de *walk* para *attack*. Na aba *Inspector*, iremos definir qual é a condição para esta transição acontecer. Vá no painel *Conditions* e adicione a condição *Attacking = true*. Clique na seta que faz a transição de *attack* para *walk* e adicione a condição *Attacking = false*.

Além disso, nas duas transições, desative os campos *Has Exit Time* e *Fixed Duration*. Coloque o valor zero no campo *Transition Duration*.

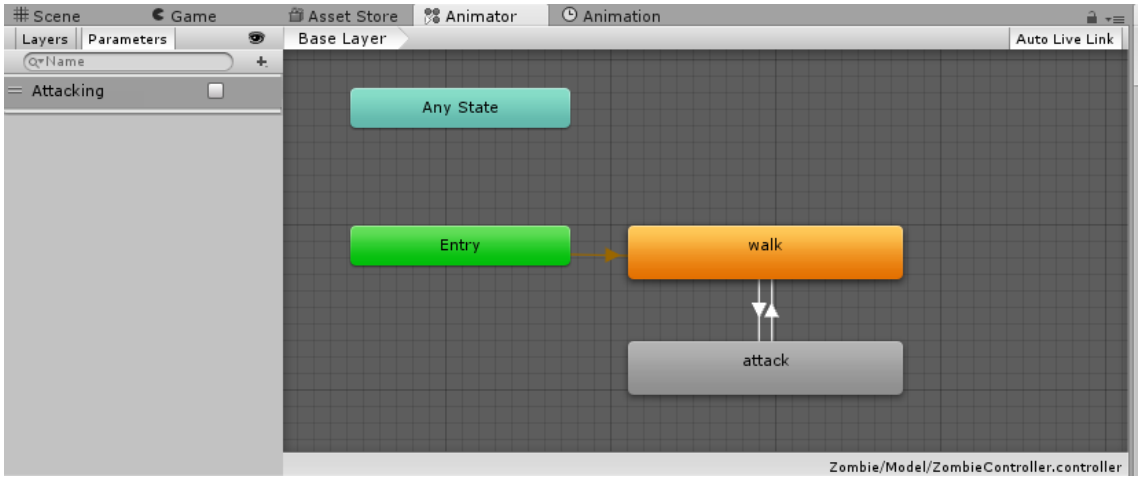


Figura 2.29. *Animator*, máquina de estados finalizada

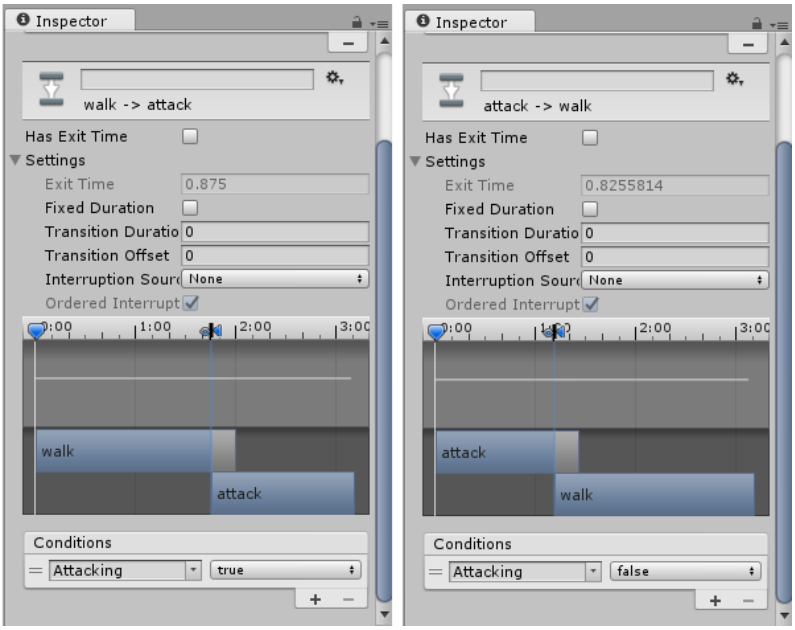


Figura 2.30. Transições

Já temos a nossa máquina de estados pronta, mas o nosso zumbi ainda não possui "inteligência" alguma, ou seja, não sabe quando deve andar, com que velocidade antes, não sabe quando está peto do jogador para atacar, etc.

Na Unity, nós temos uma maneira já pronta de fazer com que um objeto siga de um ponto de partida até um ponto de chegada, procurando a rota mais viável através de algoritmos de *pathfinding*. É dessa maneira que faremos o nosso zumbi perseguir o nosso jogador, procurando sempre o caminho mais viável, desviando de obstáculos.

Vá no painel superior e clique em *Window* > *Navigation*. Nesta nova aba, clique em *Bake*. *Bake* irá criar o que chamamos de *NavMesh*, que é uma área azul que informa quais são as áreas do cenário que são "andáveis", ou seja, não são obstáculos, buracos, etc. O processo de *bake* terá terminado quando você ver que o seu cenário ficou com partes azuladas no chão onde não houverem obstáculos.

Agora que temos uma *NavMesh* pronta, vá no objeto *Zombie* e adicione a componente *NavMeshAgent*. Esta componente é responsável por mover um objeto em uma *NavMesh*. Mude o valor do campo *Speed* para 2.

Na pasta Scripts, crie um script C# nomeado *ZombieBehaviour*. Seguiremos com a explicação do código.

```
4  public class ZombieBehaviour : MonoBehaviour {
5
6      private GameObject player;
7      private NavMeshAgent navAgnt;
8      private Animator anim;
9
10     void Awake(){
11
12         player = GameObject.FindGameObjectWithTag ("Player");
13         navAgnt = GetComponent<NavMeshAgent> ();
14         anim = GetComponent<Animator> ();
15
16     }
17
```

Figura 2.31. *ZombieBehaviour*, inicialização

Para esta classe nós teremos apenas variáveis privadas.

A variável *Player* servirá para guardar uma referência ao nosso jogador, para que o zumbi possa seguir a sua posição.

NavAgent guardará uma referência à componente *NavMeshAgent* que adicionamos ao zumbi.

Anim guardará referência à componente *Animator* que adicionamos ao zumbi.

No método *Awake*, fazemos a inicialização destas variáveis. Na linha 12, o comando *GameObject.FindGameObjectWithTag("Player")* busca um objeto na cena que possua a *tag Player*. Para isto, vá no *Unity Editor* e no painel superior vá em *Edit > Project Settings > Tags & Layers*. Escolha qualquer *User Layer* e preencha com o valor *Player*. Em seguida, selecione o objeto *First Person Controller* e, na aba *Inspector*, coloque a tag *Player* neste.

```
18     void Update(){
19
20         navAgnt.SetDestination (player.transform.position);
21
22     }
```

Figura 2.32. *ZombieBehaviour*, *Update*

Toda componente do tipo *NavAgent* possui um destino, que será o lugar para onde aquele objeto irá se mover. Como queremos que o zumbi siga o nosso personagem o tempo todo, verificando a sua posição a cada frame, fazemos isto na *Update*.

```
24      void OnTriggerEnter(Collider c){
25
26          if (c.tag == "Player")
27              anim.SetBool ("Attacking", true);
28
29      }
30
31      void OnTriggerExit(Collider c){
32
33          if (c.tag == "Player")
34              anim.SetBool ("Attacking", false);
35
36      }
37
```

Figura 2.33. *ZombieBehaviour*, *Triggers*

Para que o zumbi possa detectar quando este está perto e longe do nosso jogador, usaremos os métodos de colisão. *OnTriggerEnter* é um evento que é acionado sempre que o zumbi se colidir com algum outro objeto que tenha um colisor. As informações sobre o outro objeto são então guardadas no parâmetro passado (neste caso, a variável "c" do tipo *Collider*). Perguntamos então se o outro objeto da colisão possui a *tag Player*. Se sim, significa que o zumbi achou o jogador e deve, então, entrar no estado de ataque. Para isso, utilizamos a referência ao nosso *Animator* e mudamos o valor do parâmetro *Attacking* para *true*. Dessa maneira, acontece a transição do estado *walk* para o estado *attack*.

OnTriggerExit é acionado sempre que o zumbi deixar de estar colidindo com algum objeto. Ou seja, quando ele deixar de colidir com algum objeto que possua a *tag Player*, significa que o jogador se distanciou e, portanto, o zumbi tem que sair do estado de ataque, e o fazemos mudando o valor do parâmetro *Attacking* para false, efetuando assim a transição do estado *attack* para o estado *walk*.

```
38      public void TheyShotMe(){
39
40          Destroy (this.gameObject);
41
42      }
43
44  }
45
```

Figura 2.34. *ZombieBehaviour*, *TheyShotMe*

Por fim, temos um método que será chamado sempre que atirarem no zumbi, ou seja, sempre que o *raycast* encontrar o colisor de um zumbi. Quando isto acontecer, o nosso zumbi será destruído. Para que este método seja chamado, adicionamos as seguintes linhas de código ao nosso script *RaycastShoot*.

```
42            if (Physics.Raycast (rayOrigin, fpsCam.transform.forward, out hit, weaponRange)) {
43
44                laserLine.SetPosition (1, hit.point);
45
46                ZombieBehaviour zombieBhv = hit.collider.GetComponent<ZombieBehaviour> ();
47
48                if (zombieBhv != null)
49                    zombieBhv.TheyShotMe ();
50
51            } else {
52
53                laserLine.SetPosition (1, rayOrigin + (fpsCam.transform.forward * weaponRange));
54
55            }
```

Figura 2.35. Classe RaycastShoot, destruindo zumbis.

A execução da linha 46 faz com que seja procurado, no objeto que o *raycast* atingiu, uma componente do tipo *ZombieBehaviour*, que possui o método que destrói o zumbi. Se a componente tiver sido encontrada, ou seja, se a variável *zombieBhv* for diferente de nulo, então o método *TheyShotMe* é chamado, e assim o zumbi é destruído.

Por fim, temos a classe *ZombieSpawner* que irá fazer com que surjam vários zumbis com o passar do tempo.

```
4 public class ZombiesSpawner : MonoBehaviour {
5
6     public Transform spawnPosition;
7     public GameObject zombiePrefab;
8
```

Figura 2.36. Classe ZombieSpawner, variáveis

A variável *SpawnPosition* irá definir uma posição onde os zumbis irão aparecer. A inicialização desta variável será feita através do editor.

ZombiePrefab define qual é o objeto zumbi que iremos fazer surgir. Também será inicializada no editor.

```
9     // Update is called once per frame
10    void Start () {
11
12        InvokeRepeating ("SpawnZombie", 2, 2);
13
14    }
15
```

Figura 2.37. Classe ZombieSpawner, *Start*

No método Start, chamamos o método *InvokeRepeating*, que define um outro método que será chamado várias vezes durante o jogo, havendo sempre um intervalo de tempo definido de, nesse caso, 2 segundos.

```
16      void SpawnZombie(){
17
18          Instantiate (zombiePrefab, spawnPosition.position, new Quaternion(0, 180, 0, 0));
19
20      }
21
22  }
23
```

Figura 2.38. Classe *ZombieSpawner*, *SpawnZombie*

Este é o método que será chamado de 2 em 2 segundos. O método instancia, a cada chamada, um objeto definido pela variável *ZombiePrefab*, na posição definida por *SpawnPosition*.

Com isso, já temos o nosso jogo quase completo, faltando apenas a integração com o Oculus Rift.

2.3.7. Integração com o *Oculus Rift*

O *Oculus Rift* DK2 é composto por três peças principais: os óculos, as lentes do óculos e a rastreador de posição.

Os óculos em si vêm com diversos de cabos para que seja feita a conexão com o computador. Possui um giroscópio para que os dados de inclinação do dispositivo possam ser capturados e esta informação possa ser direcionada ao mundo virtual. Ele vem também com um par de lentes acoplado e um par de lentes extra. O rastreador de posição vem também com cabos, que deverão ser conectados tanto no computador quanto no próprio *Oculus Rift*.

Figura 2.39. *Oculus Rift, Headset*

Figura 2.40. *Oculus Rift, Tracker*

A integração que iremos fazer é bem simples. Primeiro baixar o setup do Oculus Rift na página inicial do site da Oculus.

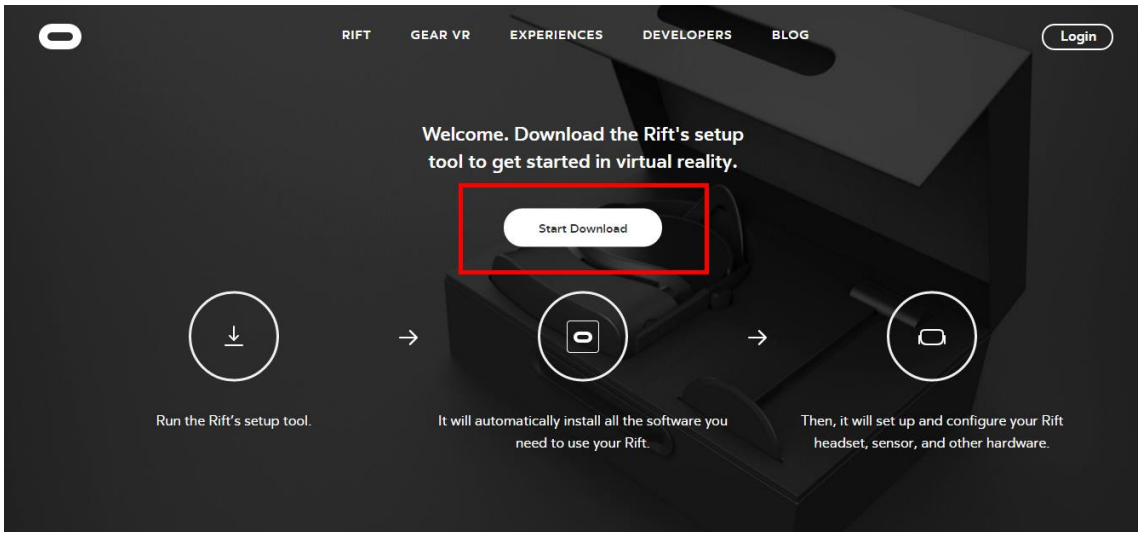


Figura 2.41. *Oculus Rift Setup*

Quando executar o setup, o seu computador já estará pronto para executar jogos feitos no *Oculus Rift*, e para desenvolver jogos também. Na página inicial da *Oculus*, vá em *Developers*, em seguida clique em *Other Downloads*, procure pelo menu *Engine Integration* e faça download do *Oculus Utilities for Unity*. A versão utilizada neste texto foi a "V1.7.0".

Com isso você terá em mãos o pacote da *Oculus*. Importe o pacote no seu projeto na Unity para ter acesso às *features*. Quando terminar de importar o pacote, dentro da Unity vá na pasta *OVR* > *Prefabs* e arraste o *asset OVRPlayerController* para a sua cena. Neste momento, você possuirá dois tipos de jogadores em cena: um que funciona sem o Oculus Rift (*FirstPersonController*) e o novo controlador, que funciona com o Rift. Delete o controlador antigo. O novo controlador também possui uma câmera, que neste caso de chama *CenterEyeAnchor*. Posicione uma pistola nesta câmera assim como foi feito antes e pronto, o jogo já está no ponto para ser jogado pelo Oculus Rift.

Além disso, no menu superior, vá em *Edit* > *Project Settings* > *PlayerSettings*. Isto irá abrir um menu na aba *Inspector*. Procure pela *checkbox* nomeada *Virtual Reality Supported* e coloque o valor como *true*. Dessa maneira, se houver algum dispositivo de realidade virtual ligada ao computador, todas as câmeras do jogo já passarão a se comportar de acordo com o dispositivo, ou seja, a leitura do giroscópio do dispositivo será feita e, baseado nesta informação, serão aplicadas rotações às câmeras do jogo e, dessa maneira, o usuário poderá "olhar em volta", no ambiente ao seu redor.

Outra funcionalidade que é acionada com o *Oculus Rift* é o de rastreamento de posição, feito pelo dispositivo *Tracker*, que deve ser posicionado no centro do monitor

onde o jogo será executado, apontando para onde o jogador estará sentado inicialmente. Com isso, o dispositivo consegue capturar algumas movimentações que o jogador faz com o *Oculus*, sendo assim possível saber quando o jogador mover a cabeça (para os lados, para frente e para trás) e, inclusive, quando este está levantado ou sentado.

2.4. Considerações finais

Neste texto foram apresentados aspectos básicos da criação de jogos na Unity. Foi desenvolvido aqui um jogo simples,

Além disso, a integração com o Oculus Rift é algo realmente simples. O passo-a-passo que foi aqui apresentado é considerando um cenário ótimo, mas deve-se considerar que é possível que um computador necessite de mais configurações para fazer o Rift funcionar. Para mais informações sobre este assunto, consultar o site da Oculus.

Referências

Unity3D (2016) “Made With Unity”, https://madewith.unity.com/games/grow-up.

Unity3D (2016) “Asset Store”, www.assetstore.unity3d.com.

Unity3D (2016) “Tutorials”, https://unity3d.com/pt/learn/tutorials/lets-try/shooting-with-raycasts?playlist=41639

Capítulo 3

Desenvolvendo Aplicações Multimídia e Ubíquas com Drones

Manoel C. Marques Neto, Rodrigo V. da Silva, Efraim Z. de Almeida Machado e Vaninha Vieira dos Santos

Abstract

Drones are platforms which can engage a set of sensors and peripherals that allow to mount an interactive multimedia environment by capturing context information, the network communication, control of other electronic devices, and others. Despite of the frequent association of drones with military defense applications, they are especially important in civil applications such as transport monitoring, communication, agriculture, disaster mitigation and environmental protection, etc. In these applications, the use of drones is always combined with multimedia capabilities. In this context, this chapter explores the use of Drones in the development of multimedia/ubiquitous applications.

Resumo

Drones são plataformas onde é possível acoplar um conjunto de sensores e periféricos que permitem montar um ambiente multimídia interativo através da captura de informação de contexto, da comunicação em rede, do controle de outros dispositivos eletroeletrônicos, dentre outros. Apesar de associação frequente de drones com aplicações de defesa militar, eles são especialmente importante em aplicações civis, como monitoramento de transporte, comunicação, agricultura, na mitigação de desastres e preservação do meio ambiente, etc. Nestas aplicações, o uso de Drones sempre é combinado com o de recursos multimídia. Neste sentido este capítulo explora o uso de Drones no desenvolvimento de aplicações multimídia/ubíquas.

3.1. Introdução

Ambientes multimídia interativos são aqueles nos quais a computação é usada para imperceptivelmente melhorar as atividades comuns. Uma das forças motrizes do interesse

emergente nos ambientes altamente interativos é tornar os computadores não apenas verdadeiramente amigáveis ao usuário, mas sim essencialmente invisíveis para ele. Para isso, um dos pontos chave é a Computação Ubíqua [Kaufmann and Buechley 2010], [Krumm 2016].

A Computação Ubíqua, em seus vários desdobramentos e aplicações, é considerada por muitos como o novo paradigma da Computação para o século XXI. É esta área da computação que estuda o acoplamento do mundo físico ao mundo da informação e fornecerá uma abundância de serviços e aplicações, permitindo que usuários, máquinas, dados, aplicações e objetos do espaço físico interajam uns com os outros de forma transparente.

O tema é considerado um dos grandes desafios da pesquisa em Computação pela National Science Foundation (NSF) [nsf] e está também presente no relatório Grandes Desafios da Pesquisa em Computação no Brasil 2006-2016, publicado pela Sociedade Brasileira de Computação (SBC) [sbc].

Drones são plataformas poderosas para aplicações ubíquas. Elas possuem interfaces necessárias para criação desde pequenos projetos ou para aqueles mais complexos. Eles usam a linguagens de programação de alto nível (por exemplo C/C++ e Java) que são é bastantes difundidas. Apesar de associação frequente de drones com aplicações de defesa militar, eles são especialmente importante em aplicações civis, como monitoramento de transporte, comunicação, agricultura, na mitigação de desastres e preservação do meio ambiente, etc. Nestas aplicações, o uso de Drones sempre é combinado com o de recursos multimídia.

Assim, este capítulo apresenta as primeiras noções de como desenvolver aplicações ubíquas com drones através de exemplos práticos com o uso de sensores. Esses conceitos introdutórios são um importante alicerce para uma gama de aplicações em áreas distintas.

O restante deste capítulo é organizado da seguinte forma: a Seção 2 contextualiza a computação ubíqua mostrando suas principais definições, princípios e tecnologias. Questões que representam desafios para construção de aplicações multimídia e ubíquas como: consumo de energia, volatilidade e comunicação entre dispositivos, testes de sistemas, entre outras são abordadas também são abordadas. A Seção 3 detalha a plataforma utilizada como base para este capítulo, listando suas características físicas e princípios físicos. Além disso, são apresentados casos de sucesso no uso de drones associado aplicações multimídia e ubíquas. A Seção 4 detalha o ambiente necessário para desenvolver aplicações multimídia e ubíquas usando drones. Nesta seção são apresentados as ferramentas e componentes básicos usados. Além disso, funcionalidades básicas sobre a IDE utlizada para desenvolvimento na linguagem Java são abordadas. A Seção 5 apresenta a estrutura básica de um programa escrito para um drone. Também são apresentadas as estruturas básicas de controle como: funções decolar, aterrizar, capturar imagem e vídeo, entre outras. A Seção 6 aborda o uso de sensores na obtenção de informação do mundo real. Por fim, a Seção 7 apresenta o resumo do que foi abordado neste capítulo numa visão crítica sobre as possibilidades abertas a partir do uso das tecnologias apresentadas no desenvolvimento de aplicações ubíquas e multimídia, apresentando também as perspectivas para este mercado.

3.2. Computação Ubíqua: definição, princípios e tecnologias

A Computação Ubíqua, em seus vários desdobramentos e aplicações, é considerada por muitos como o novo paradigma da Computação para o século XXI. É esta área da computação que estuda o acoplamento do mundo físico ao mundo da informação e fornecerá uma abundância de serviços e aplicações, permitindo que usuários, máquinas, dados, aplicações e objetos do espaço físico interajam uns com os outros de forma transparente. O tema é considerado um dos grandes desafios da pesquisa em Computação pela National Science Foundation (NSF) e está também presente no relatório Grandes Desafios da Pesquisa em Computação no Brasil 2006-2016, publicado pela Sociedade Brasileira de Computação (SBC).

A computação ubíqua é um termo utilizado para expressar a informática de forma onipresente no cotidiano. A ideia da computação ubíqua é que a computação será onipresente, estendendo os limites da computação para fora dos computadores, tornando-se pervasiva no cotidiano, de forma "invisível". A invisibilidade da computação úbiqua não significa que ela não é visível, mas sim que as interfaces entre as pessoas e os dispositivos eletroeletronicos utilizados sejam as mais naturais possíveis, de forma que as pessoas não percebam que estão interagindo com um dispositivo. Por causa deste objetivo, a computação úbiqua envolve a análise e estudo das interfaces naturais de comunicação humana, como fala, gestos, movimentação do globo ocular dentre outros, e o estudo da computação sensível ao contexto, de forma que as aplicações úbiquas possam capturar automaticamente o contexto do usuário e alterar, suavente, sua forma de interagir com o mesmo. Assim, podemos citar as principais características da computação úbiqua:

- Diversidade e onipresença de dispositivos de forma a abranger a maioria dos ambientes;
- Descentralização do processamento de forma que cada dispositivo (ou grupos de dispositivos) seja resposável por analisar seus dados e executar suas atividades;
- Conectividade dos dispositivos de forma que eles possam trocar informações entre si para conseguir obter uma maior precisão na detecção do contexto;

Na computação úbiqua, há uma mudança na interação homem-máquina pois, o homem passa a interagir com a máquina de maneira passiva pois, a máquina deixa de ser o foco das atenções. Apenas o resultado obtido pela máquina interessa, de forma que esta interação homem-máquina aconteça de forma imperceptiva e suave, o que é chamado de *Calm Technology*.

A computação sensível ao contexto requer a utilização de sensores para obter informações do ambiente para identificar o contexto do usuário de forma mais precisa. A necessidade de ser "invisível" da computação úbiqua requer dispositivos eletroeletronicos pequenos e baratos, que possam ser distribuídos e conectados à internet ou a outros dispositivos através de redes sem fio. Estas duas condições definem que as tecnologias utilizadas para a computação úbiqua são as tecnologias da computação móvel e da compução pervasiva pois, consistem em dispositivos que possuem as seguintes características:

- computadores móveis: computadores móveis, pequenos, que gerenciam recursos de energia de forma eficiente;
- computadores wearable e interfaces hands-free: computadores wearable são computadores projetados para o uso sem a necessidade das mãos, que utilizam interfaces hands-free, isto é, não há a predominância das mãos como fornecedora principal de informação, utilizando sensores (câmera e microfones) para tal;
- conexões wireless: a maioria dos dispositivos móveis (computadores móveis e computadores wearable) possuem conexões sem fio com a internet;
- consciência do contexto: utilização do contexto de forma que o aplicativo se adpate a condição atual do usuário;
- ambientes inteligentes: utilização de ambientes inteligentes, que possuem comportamentos automáticos ativados por determinados acontecimentos sem nenhuma instrução explícita do usuário.

Das características da computação úbiqua e de suas tecnologias, podemos citar características das aplicações úbiquas:

- Computação Desagragada e Sensível a Posição: onde há uma reconfiguração dos dispositivos de interface, de forma a exibir a informação de acordo com o posicionamento do usuário;
- Realidade Aumentada: computadores móveis ou wearables são combinados de sensores de forma a trazer ao usuário uma superposição da realidade e dos dados cálculados por estes dispositivos;
- Link entre Objetos Físicos e Abstratos: há um relacionaamento entre um objeto físico do mundo real e um objeto abstrato, como por exemplo uma webpage.

Estas características da computação úbiqua trazem alguns desafios, como por exemplo:

- Privacidade: Como manter a privacidade do usuário em meio a uma quantidade enorme de sensores e dados do usuários armazenados para detecção de contexto;
- Complexidade: como manter as aplicações fáceis de utilizar a medida que as coisas acontecem automaticamente e o usuário perde o controle do que a aplicação faz;
- Consumo de energia: como manter os pequenos dispositivos funcionando por mais tempo visto que o seu tamanho e seu poder de processamento influenciam diretamente na sua capacidade de armazenar energia e de consumir, respectivamente;
- Testes de aplicativos: como testar aplicativos que têm suas funcionalidades ativadas a depender do contexto;

Considerando os objetivos, as características e tecnologias da computação úbiqua, a seção seguinte explora o drone como uma plataforma para a construção e aplicações úbiquas.

3.3. Drones como Plataforma de Aplicações Multimídia

Um veículo aéreo não-tripulado (UAT), também conhecido como drone, é qualquer aeronave que não necessita de um piloto embarcado para ser controlado. Este tipo de aeronave é controlado à distância por meios eletrônicos e computacionais, sob a supervisão de humanos, ou mesmo sem a sua intervenção. Drones foram inicialmente projetados para guerra, para realizar missões arriscadas para humanos. Atualmente, os drones possuem diversos usos, fora da área militar, como por exemplo, fotografias e imagens áreas de eventos ou monitoramento de áreas de desastres, ou são usados em fazendas modernas para otimizar a pulverização de fertilizantes e pesticidas [**?**].

3.3.1. Funcionamento Básico do Drone

Existem diversas categorias de drones, a depender de seu uso e,, neste capítulo abordamos drones comerciais usados por cívis para entretenimento que podem ser controlados por um computador (móvel ou não). Estes drones possuem tecnologia suficiente para serem utilizados em diversas aplicações práticas e úteis no dia-a-dia, possuindo uma autonomia de voo menor que 20 minutos e um alcançe de controle limitado (menor que 1 km). As principais partes destes drones são:

- Corpo: que constitui a maior parte do drone. Geralmente formado por quatro hélices rotatórias;
- Bateria: responsável por armazenar a energia que será utilizada no drone. Geralmente as baterias dos drones são compostas de polímero de lítio;
- Atuadores: são os meios físicos responsáveis por controlar o drone, como controladores de velocidade, motores, propelentes, leds, etc;
- Controlador de vôo: é o sistema de hardware do drone responsável por interagir com os comandos do controle e controlar os atuadores;
- Sensores: são responsáveis por detectar o estado do drone, o que pode ser entendido como o contexto do drone. Alguns exemplos de sensores são o posicionamento GPS, medidores de distância, et;
- Software: responsáveis por processar os dados obtidos dos sensores e utilizar o controlador de voo para utilizar os atuadores, mantendo o estado do drone de acordo com o desejado. Um exemplo desta ação é quando o drone evita colidir com os obstáculos. O software utiliza informações dos sensores de distância, e de acordo com os dados obtidos, avisa o controlador de voo para mover-se na direção contrária do obstáculo. Quando há um usuário pilotando o drone, o software age de forma secundária, isto é, o controle do drone é realizado pelo usuário.

Assim, podemos perceber que de forma resumida, o funcionamento do drone consiste em o controlador de voo receber comandos (de um software de forma autônoma ou através de um usuário) e utilizar os atuadores para alterar o estado do drone de acordo com o comando. Durante esta tarefa, os sensores, automaticamente, captam informações do meio.

Independente do comando passado para o controlador de voo, estes comandos podem ser resumidos em uma movimentação do drone sobre as três dimensões X, Y e Z, projetadas sobre o drone. O eixo X é uma linha imaginária que passa pela frente e fundo do drone, o eixo Y passa de forma ortogonal ao eixo X, indo de um lado ao outro do drone e o eixo Z é ortogonal aos eixos X e Y e indo da parte superior do drone a parte inferior. O que faz com o que o drone se desloque em cada uma destas três dimensões é a inclinação que o drone faz com cada um dos eixos. Estas inclinações são chamadas de Roll (inclinação com o eixo X), Pitch (inclinação com o eixo Y) e Yaw (inclinação com o eixo Z). A Figura 1.1 exibe um drone, os eixos e as três inclinações.

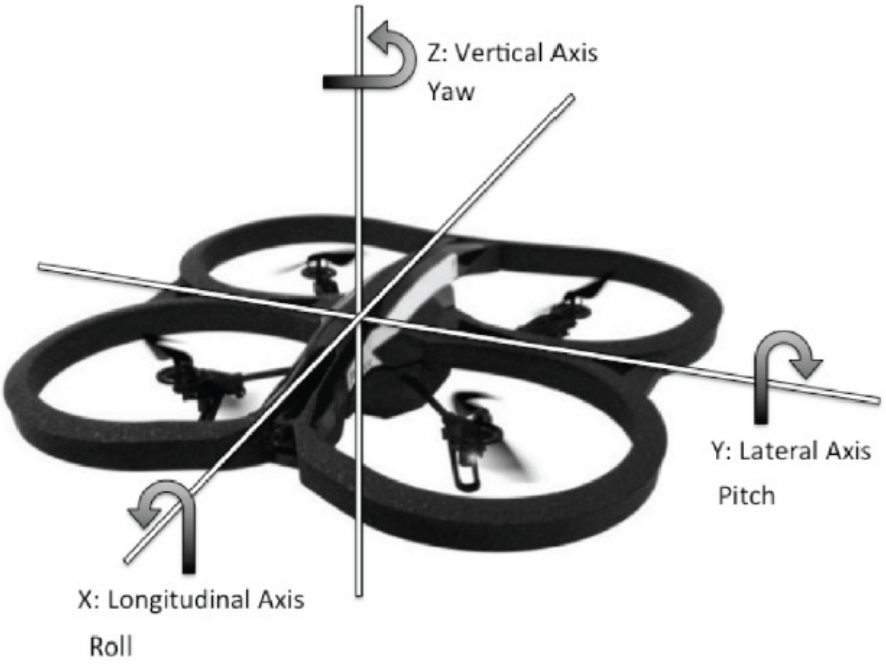


Figura 3.1. Eixos drone ??

A combinação de sentido e força de rotação de cada hélice impulsiona o drone de forma que, combinado com a inclinação do drone faz com que ele se desloque sobre os três eixos. Assim, quando um software ou usuário informa ao controlador de voo que o drone deve ir para frente, o controlador de voo converte este comando em uma determinada inclinação do drone e uma determinada potência para cada motor. Os drones atuais utilizam diversas rotinas secundárias para estabilizar o voo do drone, de forma que o comando dado aos atuadores pelo controlador de voo leva em consideração não só os comandos vindos do usuário ou software, mas também estas rotinas de estabilização. Além destas rotinas, drones mais modernos também têm interfaces simples para comandos complexos, de forma a ter um maior nível de abstração sobre rotinas comuns nas mais diversas aplicações que utilizam drones:

- Self-Level: estabilização automática da altitude do drone;
- Hover: estabilização automática dos ângulos dos eixos *pitch*, *roll* e *yaw*;

- Care-free: controle automático sobre os eixos *rol*l e *yaw* enquanto o drone se move horizontalmente;
- Pouso e decolagem: fazem o drone levantar voo ou pousar automaticamente;
- Failsafe: automaticamente pousa o drone se ele perder sinal do controlador;
- Return-to-home:
- Follow-me: função que faz o drone seguir o controlador através do seu posicionamento GPS;
- Navegação por pontos GPS: permite determinar uma rota para o drone através de uma série de pontos GPS;
- Truques pré programados: funcionalidades que permitem a realização de manobras aéreas complexas como giros e *loops* com um simples comando;

3.3.2. Drones e Aplicações Multimídia

Até o momento apenas analisamos o funcionamento básico do drone, todavia drones também vem equipados com sensores, como por exemplo, com camêras, termômetros e outros sensores que podem ter seus dados enviados do drone para um computador (ou computador móvel, como mostraremos a seguir) com um poder maior de processamento (e uma flexibilidade maior de configuração) e neste computador estes dados podem ser analisados de forma automática e convertidos em determinados comandos enviados ao drone novamente. Assim, é possível definir de forma autônoma o comportamento do drone de acordo com os dados processados. Por exemplo, é possível utilizar processamento de imagens obtido pelo drone para fazer com que ele acompanhe uma determinada pessoa ou objeto, por exemplo.

Assim, drones são uma poderosa plataforma de desenvolvimento de aplicações úbiquas visto que eles consistem, de maneira genérica, em um computador móvel, repleto de sensores (câmera, gps, sensores de pressão, sensores de velocidade, acelelômetros, dentre outros) e livres de restrições de movimentos (drones voam) que podem ser utizados como uma extensão física do usuário pois, possibilitam uma integração entre o usuário e meio de uma forma não anteriomente permitida, abrindo espaço para obtenção de uma enorme quantidade de dados, o que abre uma gama de aplicações possíveis.

3.4. Ambiente de Desenvolvimento em um Drone

Considerando a ampla gama de drones existentes no mercado, para desenvolver uma aplicação para drones, é necessário escolher um drone que possua as características que satisfaçam a necessidade da aplicação a ser desenvolvida. Claramente, uma destas característica é que o drone possua um API ou SDK disponível. Outras características que devem ser levadas em consideração dependem do domínio da aplicação como tempo de voo ou a existência de um determinado sensor. Assim, analisando diversos fabricantes, escolhemos o drone Phantom 4 da DJI porque ele possui um SDK para desenvolvimento de aplicativos desktops e móveis.

Assim, após a decisão de qual drone utilizar, o próximo passo é decidir se a aplicação será desktop ou móvel. Iremos mostrar como desenvolver um aplicativo móvel para o drone Phantom 4. Escolhemos a plataforma móvel android por esta representar uma grande fatia dos dispositivos móveis no mercado. Assim, após escolher as plataformas-alvo da nossa aplicação (android - versão 4.4 KitKat ou superior - e drone Phantom 4 da DJI), o próximo passo é configurar a IDE de desenvolvimento.

3.4.1. Android Studio

O Android Studio é a IDE oficial de desenvolvimento de aplicações para Android, ela oferece funcionalidades e ferramentas para facilitar a criação de interface gráfica, aumentar a produtividade na escrita de código, avaliação de performance da aplicação, entre outros. A IDE possui inúmeras funcionalidades interessantes, porém neste curso será abordado apenas o básico para a criação de uma aplicação Android para Drones.

Toda informação utilizada nesta seção pode ser encontrada na página oficial do Google [Google Android Studio Guide].

3.4.1.1. Criação de projeto

Para a criação do projeto é preciso abrir o Android Studio e selecionar a opção '*Start a new Android Studio project*', conforme exibido na Figura 1.2

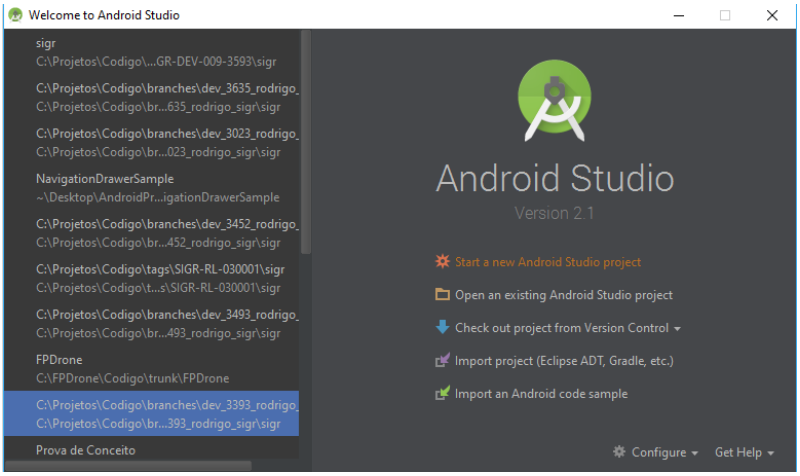


Figura 3.2. Criação do Projeto no Android Studio

A tela seguinte, de título '*New Project*', possibilita a configuração das informações iniciais do aplicativo:

- *Application name* é o nome da aplicação;
- *Company Name* é o domínio da empresa dona da aplicação;
- *Package Name* é o nome do pacote raiz da aplicação, geralmente ele é formado pelo domínio da empresa junto com o nome da aplicação, mas é possível customizar este

nome clicando em 'Edit' no canto direito. Este nome do pacote será utilizado para registro da aplicação no site da DJI;

- *Project location* é o local onde os arquivos do projeto serão mantidos.

A Figura 1.3 apresenta uma possível configuração inicial para exemplo do curso.

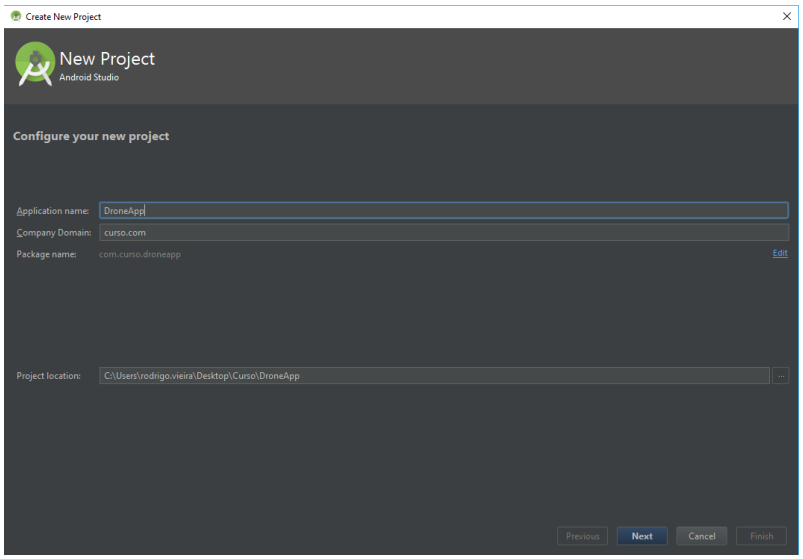


Figura 3.3. Configuração do nome da aplicação e pacote

Na Tela '*Target Android Devices*' pode-se selecionar para quais dispositivos e versões a aplicação será desenvolvida, a DJI recomenda que a aplicação seja desenvolvida para versões acima da KitKat, API 19. Neste curso não serão abordadas versões para vestíveis, TV ou Automóveis, apenas para smartphones.

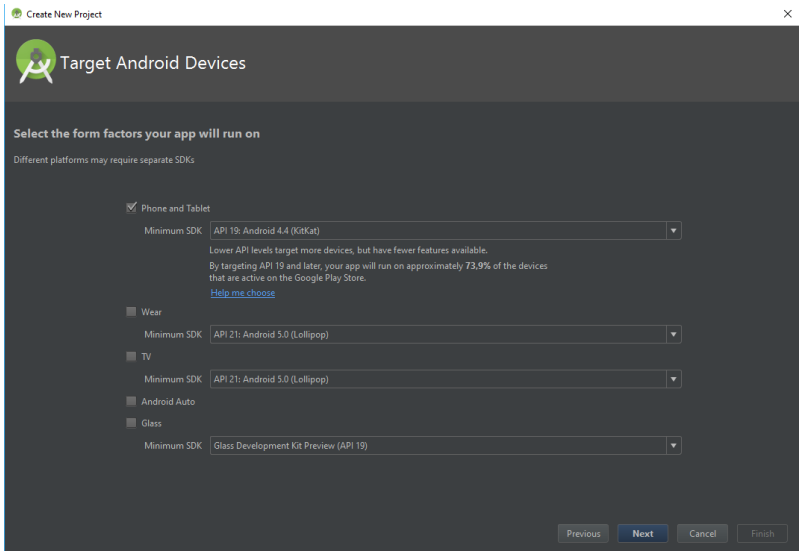


Figura 3.4. Configuração de dispositivos alvo

Na etapa seguinte, como mostrado na Figura 1.5, o objetivo é selecionar a configuração e layout da primeira tela da aplicação. O Android Studio disponibiliza diversas telas pré-desenvolvidas que facilitam o aprendizado, para a criação da aplicação de exemplo do curso será utilizada uma tela vazia (*Empty Activity*), e então a aplicação será construída passo a passo.

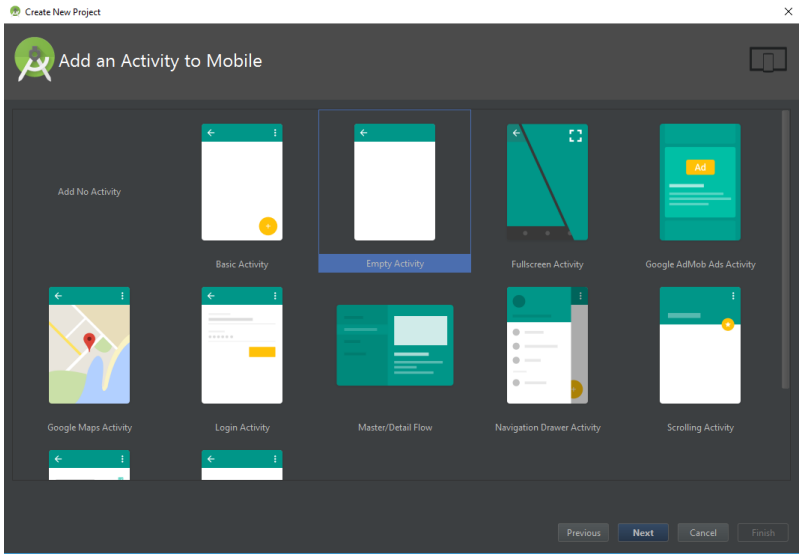


Figura 3.5. Seleção da primeira tela

Selecionado o tipo de tela, agora devemos dar nome à tela e ao arquivo xml de layout, neste exemplo será mantido o nome padrão sugerido pelo Android Studio, *Main Activity*.

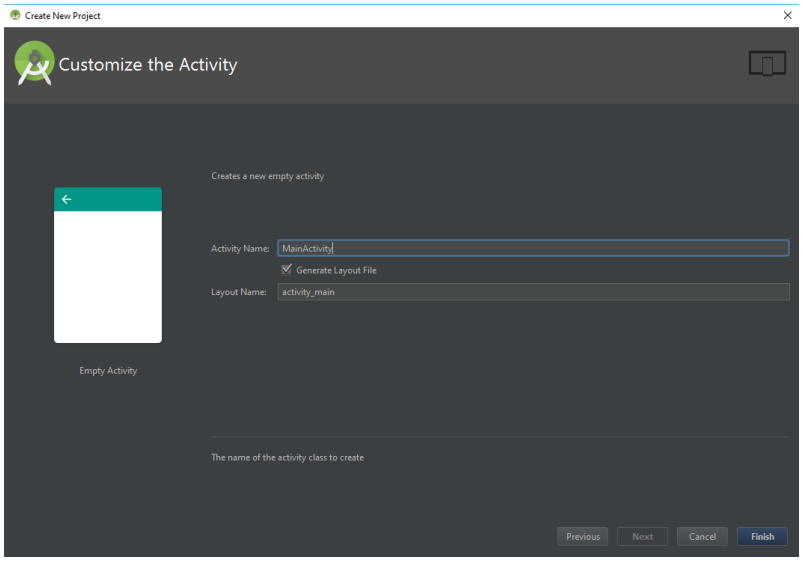


Figura 3.6. Configuração do nome da primeira tela

Com o projeto de aplicação criado, o Android Studio apresenta sua interface de

visualização do projeto. À esquerda tem-se à disposição todos os arquivos do projeto, à direita o editor de texto e interface e na parte superior se encontram as barras de ferramentas.

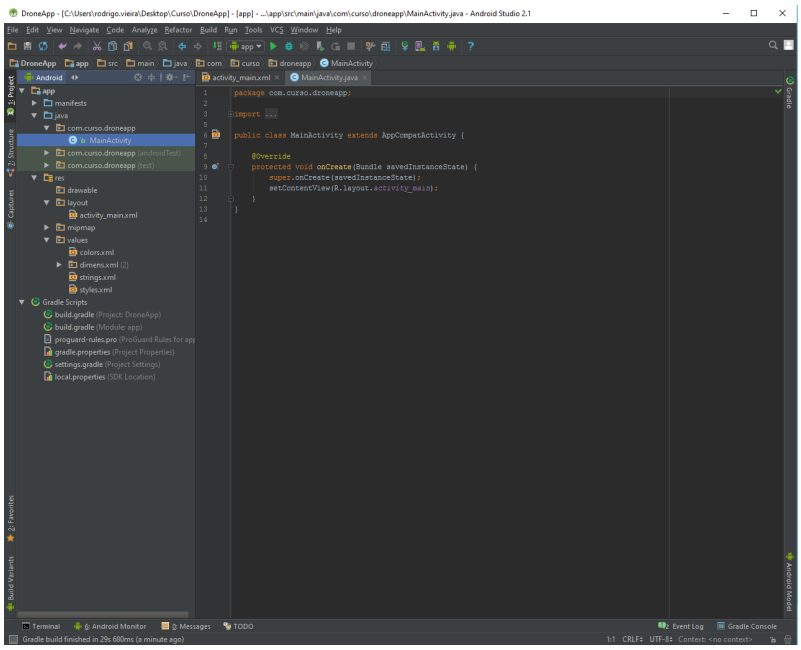


Figura 3.7. Tela inicial do projeto no Android Studio

3.4.1.2. Criação de arquivos e classes

Para criar um arquivo ou classe no Android Studio existem duas formas de proceder: (1) na barra de ferramentas do Android Studio seleciona-se *File -> New -> [classe, pasta, pacote, arquivo xml, etc.]*; ou (2) pode-se clicar com o botão esquerdo na pasta ou pacote onde o arquivo deve ser salvo, e selecionar *New -> [classe, pasta, pacote, arquivo xml, etc.]*

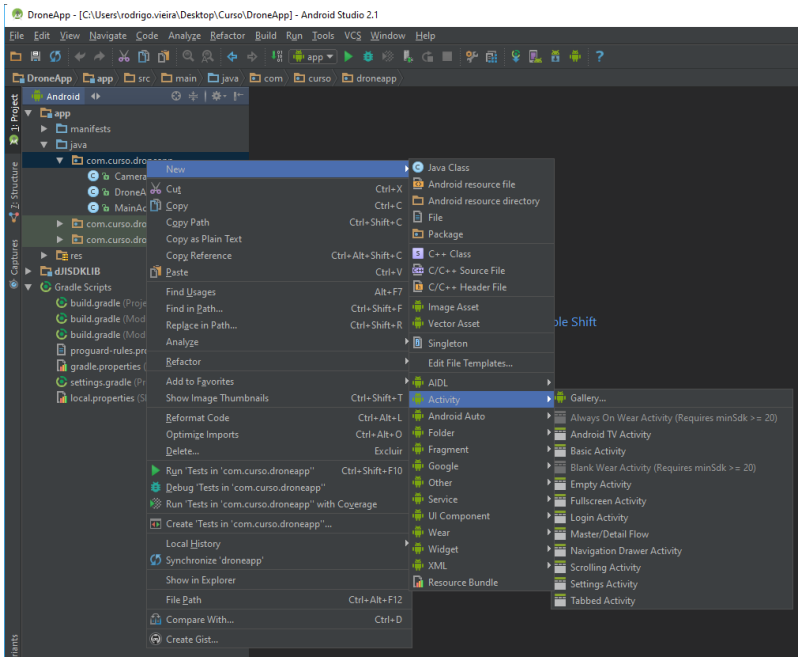


Figura 3.8. Criação de arquivos e classes no Android Studio

Uma funcionalidade importante do Android Studio é a possibilidade de utilização de telas pré-desenvolvidas. A IDE oferece telas de login, configurações, com menu lateral, com barra de rolagem, abas, entre outras opções. As classes pré-desenvolvidas vêm com comentários explicativos, o que auxilia no processo de aprendizado.

3.4.1.3. Construção de interface gráfica

No Android Studio existe a possibilidade de construção de interface gráfica de duas formas:

Pode-se arrastar os componentes com o mouse, posicioná-los na tela e alterar suas configurações via interface gráfica, conforme apresentado na Figura 1.9. No menu à esquerda estão disponibilizados todos os componentes de interface padrão do Android, no menu à direita estão as configurações referentes ao componente selecionado.

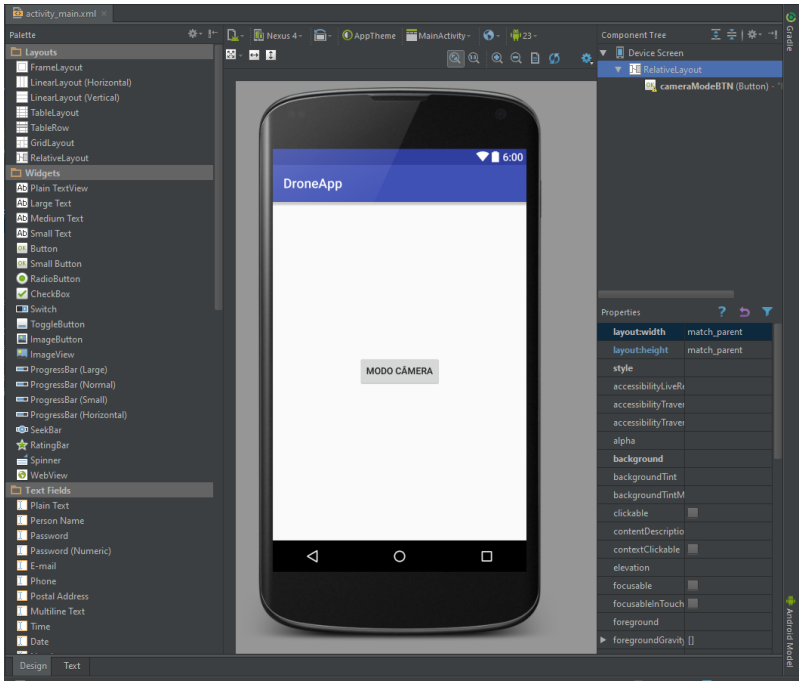


Figura 3.9. Construção de interface gráfica

Ou pode-se adicionar os componentes via texto diretamente no arquivo xml, conforme apresentado na Figura 1.10. Todas as alterações realizadas no editor de texto são refletidas em tempo real na previsualização localizada na lateral direita.

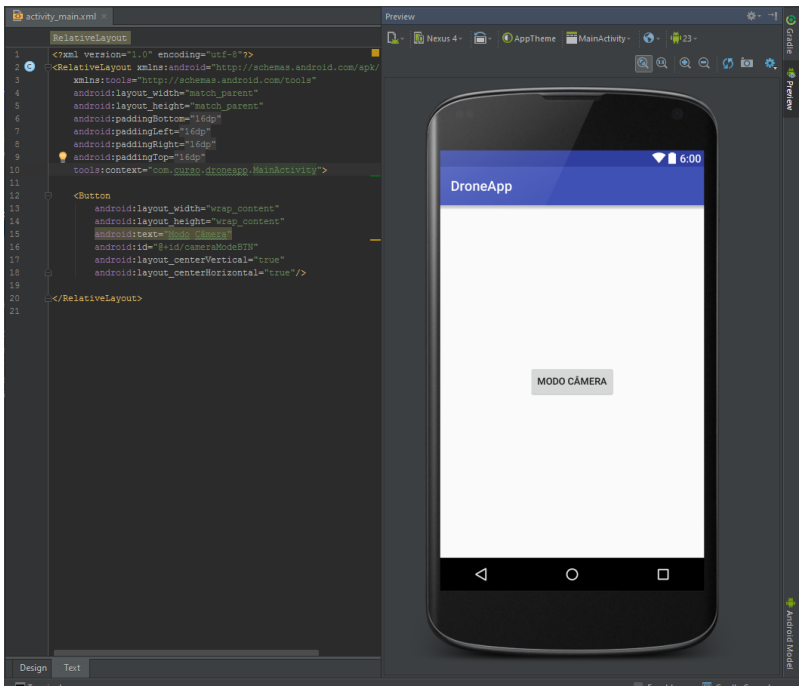


Figura 3.10. Construção de interface gráfica via texto

3.4.1.4. Execução e depuração uma aplicação

Para executar a aplicação seleciona-se *Run -> Run 'app'*, ou o ícone triangular verde da barra de ferramentas, conforme apresentado na Figura 1.11.

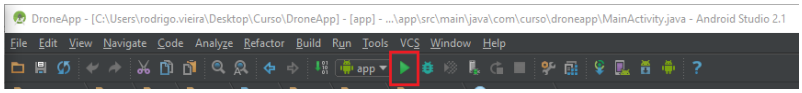


Figura 3.11. Execução da aplicação

Então a IDE abre um popup para que se possa selecionar em qual dispositivo a aplicação deve ser executada, conforme apresentado na Figura 1.12.

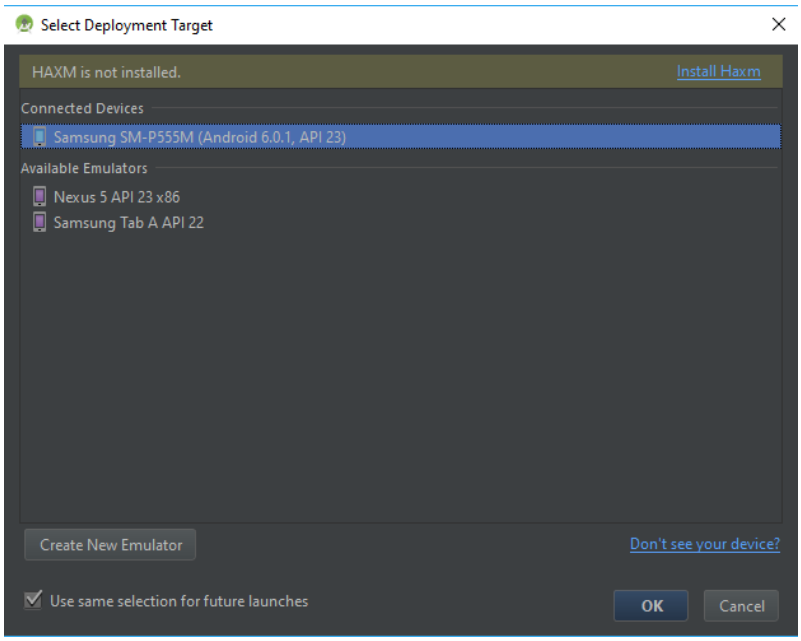


Figura 3.12. Seleção do dispositivo para execução da aplicação

Para debugar a aplicação breakpoints podem ser adicionados no código através de clique no espaço entre o editor de texto e os arquivos do projeto, um círculo vermelho marca a posição do breakpoint, conforme apresentado na Figura 1.13.

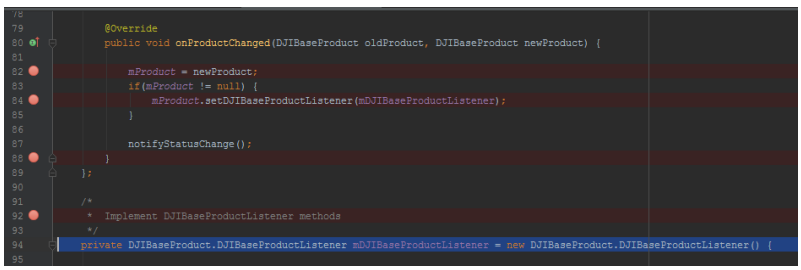


Figura 3.13. Marcação de Breakpoints

A aplicação pode ser executada em modo debug via clique no ícone em formato de 'inseto' na barra de ferramentas, conforme apresentado na Figura 1.14, então o popup ilustrado na figura 1.13 aparece novamente para seleção do dispositivo de execução.

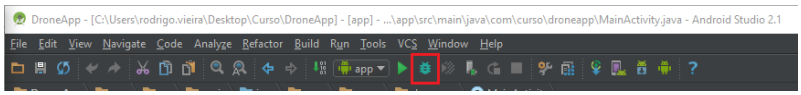


Figura 3.14. Execução da aplicação em modo Debug

Através da aba Debug é possivel passar passo a passo pelo código e inspecionar as variáveis:

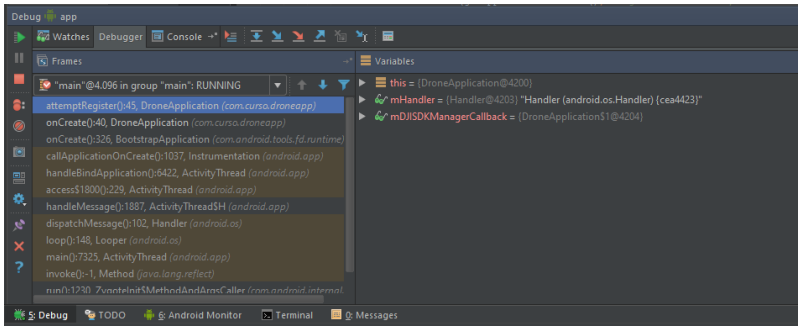


Figura 3.15. Aba de Debug

3.4.2. Kits de desenvolvimento da DJI

Este curso abordará o desenvolvimento de aplicações móveis especificamente para Drones da marca DJI, já que a empresa oferece todo um suporte ferramental que facilita o desenvolvimento de aplicações para seus dispositivos de voo. A DJI desenvolveu três kits de desenvolvimento para seus Drones :

- Onboard SDK, é o kit de desenvolvimento que oferece funcionalidades para desenvolvimento de software para o firmware do Drone. Utilizando este SDK é possível desenvolver aplicações que podem transformar o Drone em um robô autônomo. Um exemplo de aplicação utilizando este SDK seria um software de captura programada de imagens que poderia levar em consideração a data e hora de programação do voo, geolocalização e transmissão ao vivo das imagens para algum orgão receptor [DJI Onboard SDK];

- Guidance SDK facilita o desenvolvimento de aplicaçoes baseadas na visão do Drone, oferencendo funcionalidades básicas para identificação de padrões a partir das imagens capturadas pelo gadget de nome Guidance, que possui hardware específico para detecção de formas a partir de câmeras e sensores a laser. Uma aplicação de mapeamento de terreno é um exemplo do que pode pode ser implementado com a utilização deste SDK [DJI Guidance SDK];

- Mobile SDK é o kit desenvolvido para facilitar o desenvolvimento de aplicações móveis capazes de se comunicar com o Drone. Com ele pode-se criar um app Android ou IOS que se comunique com o Drone e utilize de todos os seus sensores para alguma finalidade [DJI Mobile SDK].

Sob o pretexto de melhor difusão da informação e facilidade de aprendizado, neste capítulo será abordada apenas a construção de aplicações móveis criadas sob a plataforma de desenvolvimento Android, utilizando o Mobile SDK da DJI e a IDE Android Studio.

3.5. Construção do Primeiro Programa

Nossa primeira aplicação será chamada DroneApp, uma aplicação que exibe o vídeo capturado pela câmera do Drone e possibilita a captura de fotos. Para criar nossa primeira aplicação, é necessário criar o projeto no Android Studio, integrar a DJI Android SDK ao projeto no Android Studio, registrar a aplicação na página da DJI, registrar a aplciação no Android Studio e finalmente criar as classes de nossa aplicação.

3.5.1. Criação o Projeto no Android Studio

Para a construção da aplicação de exemplo deste curso foi criada uma pasta na área de trabalho chamada *Curso*, onde serão mantidos os arquivos necessários para a construção do projeto de aplicação Android.

O projeto no Android Studio foi criado conforme explicado na seção 1.4.1.1

3.5.2. Integração do DJI Android SDK ao Projeto no Android Studio

As informações utilizadas nesta subseção foram obtidas do tutorial oficial da DJI [DJI Workflow Integrate].

É preciso fazer Download do DJI Android SDK no site oficial da ferramenta [DJI Mobile SDK] e extrair os arquivos para dentro da pasta *Curso*, na área de trabalho.

Para utilizar as funcionalidades do Android SDK da DJI deve-se importar o SDK como um módulo da aplicação no Android Studio. Para isso seleciona-se *File -> New -> Import Module* na barra de ferramentas do Android Studio, conforme apresentado na Figura 1.16.

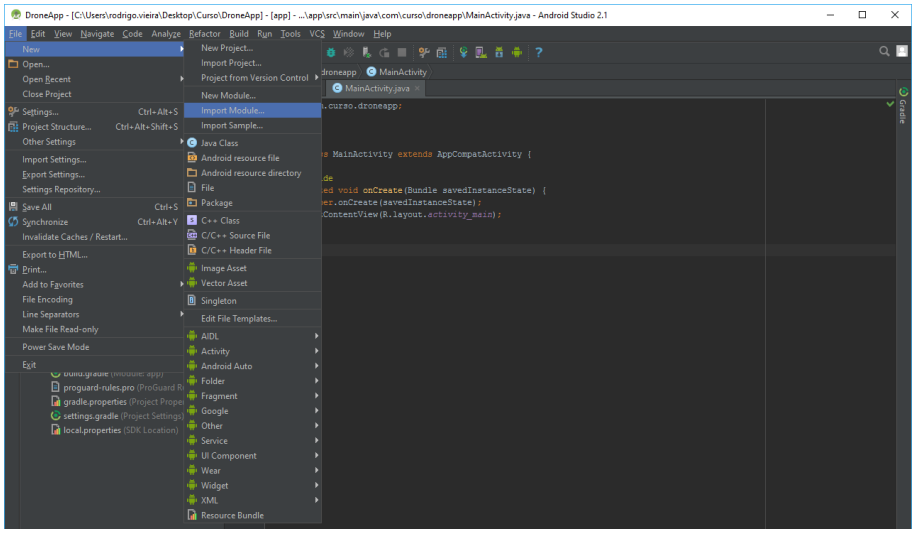


Figura 3.16. Importação do múdulo

Na tela seguinte deve-se selecionar a pasta *API Library* dentro do diretório do DJI Android Mobile SDK. No case desta aplicação de exemplo, essa pasta está localizada no diretório Curso, na Área de Trabalho. conforme apresentado na Figura 1.17.

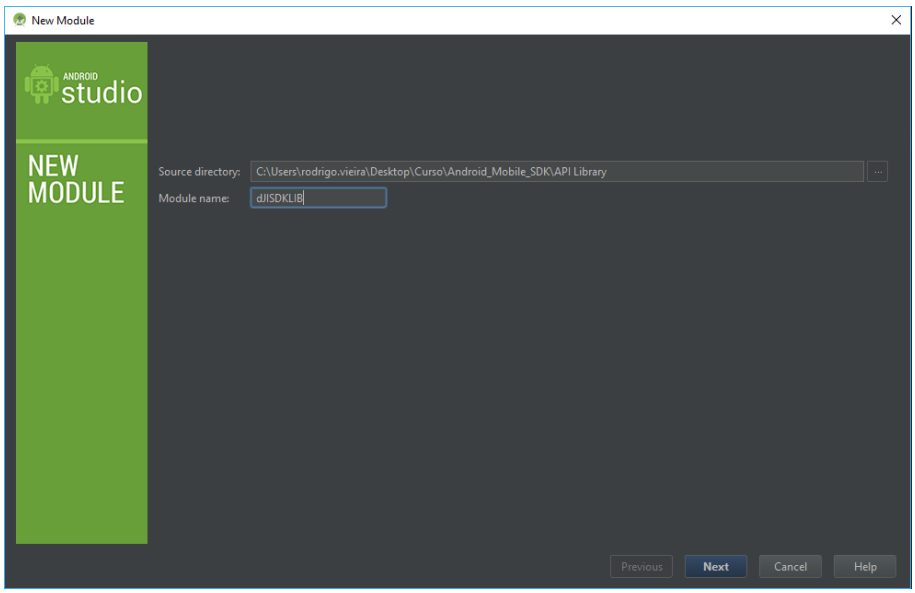


Figura 3.17. Configuração do nome do módulo e local do diretório fonte

A próxima tela exibe algumas opções de configuração de importação do módulo, porém não será preciso realizar alterações para este evemplo. O botão '*Finish*' finaliza o processo de importação. conforme apresentado na Figura 1.18.

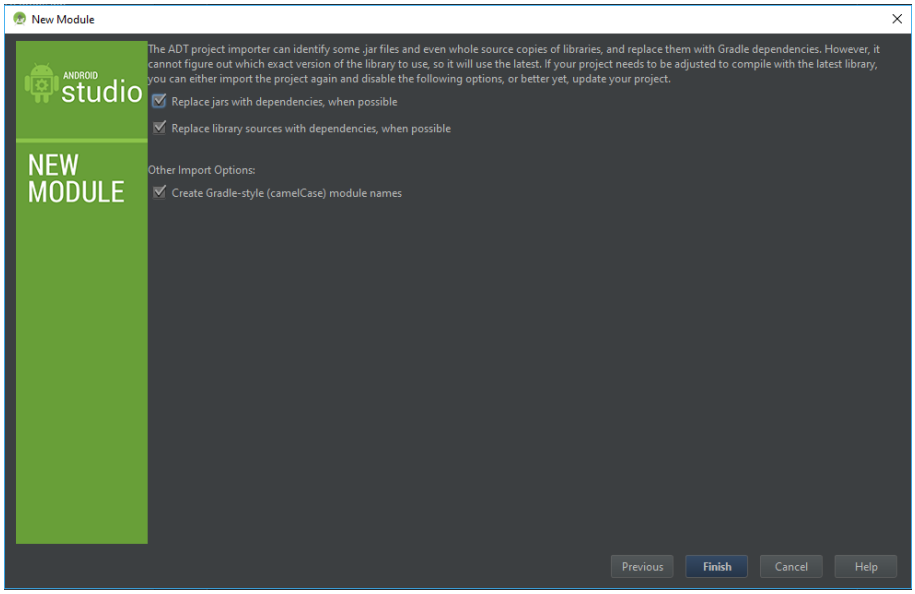


Figura 3.18. Finalização da importação do módulo

O Android Studio deve, automaticamente, compilar o projeto para integrar o módulo recém importado. Ao fim da tarefa, o arquivo de configuração do gradle, módulo app, deve ser aberto para alterações, conforme apresentado na Figura 1.19.

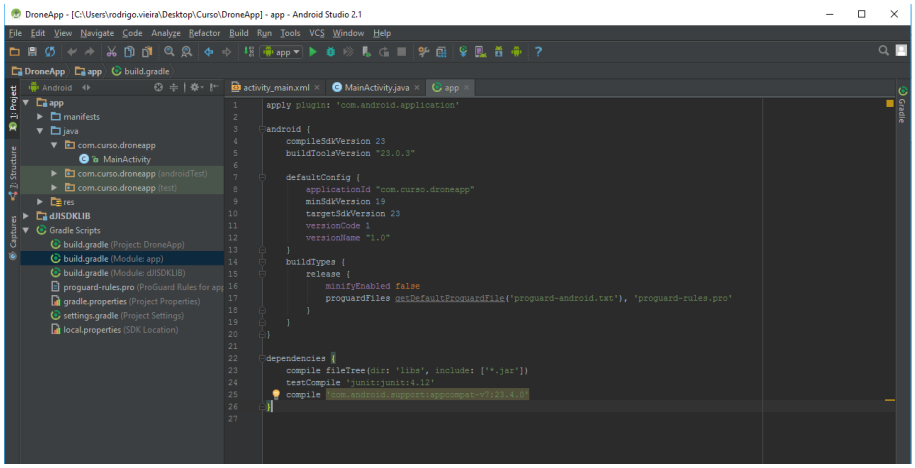


Figura 3.19. Arquivo de configuração Gradle

Adiciona-se o texto '*compile project(':dJISDKLIB')*' à seção *dependencies* do arquivo de configuraçção, conforme figura 1.21. Essa adição informará ao Android Studio que o módulo djiSDKLIB deve ser compilado juntamente com a aplicação que será construída.

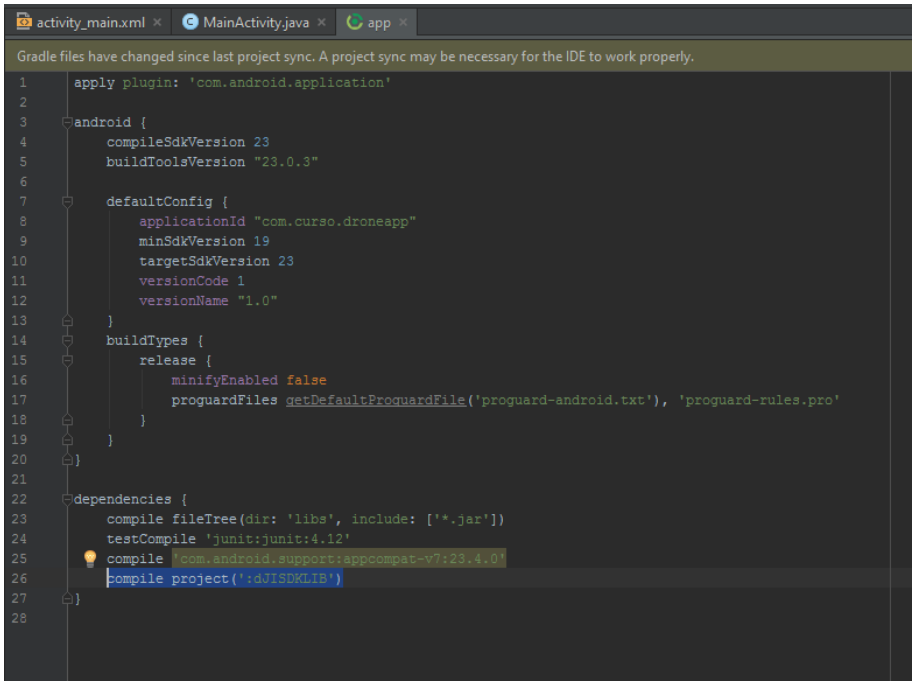


Figura 3.20. Arquivo de configuração do Gradle com adição do módulo importado

Algumas bibliotecas que o SDK da DJI precisa não são configuradas automaticamente, portanto precisa-se abrir o arquivo de configuração Gradle do módulo djiSDKLIB, conforme figura 1.21.

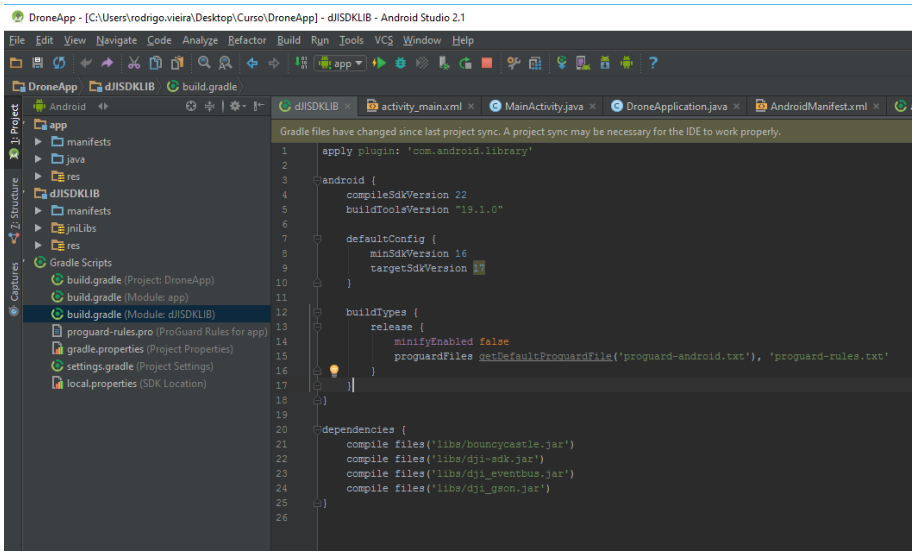


Figura 3.21. Arquivo de configuração do Gradle do módulo djiSDKLIB

A tag dependencies precisa ser alterada da seguinte forma:

```
dependencies {
  compile 'com.google.code.gson:gson:2.6.2'
  compile files('libs/converter-gson-2.0.1.jar')
```

```
    compile files('libs/dji-sdk.jar')
    compile files('libs/dji_eventbus.jar')
    compile files('libs/dji_gson.jar')
    compile files('libs/okhttp-3.2.0.jar')
    compile files('libs/okio-1.7.0.jar')
    compile files('libs/retrofit-2.0.1.jar')
    compile files('libs/rxandroid-1.1.0.jar')
    compile files('libs/rxjava-1.1.2.jar')
    compile files('libs/sqlbrite-0.6.2.jar')
}
```

Agora basta sincronizar o projeto de aplicação com as novas configurações do Gradle, para que sejam descarregados os arquivos .jar necessários para a criação da aplicação. Na barra de ferramentas do Android Studio a opção *Tools -> Android -> Sync Project with Gradle Files* sincroniza a configuração, conforme apresentado na Figura 1.22.

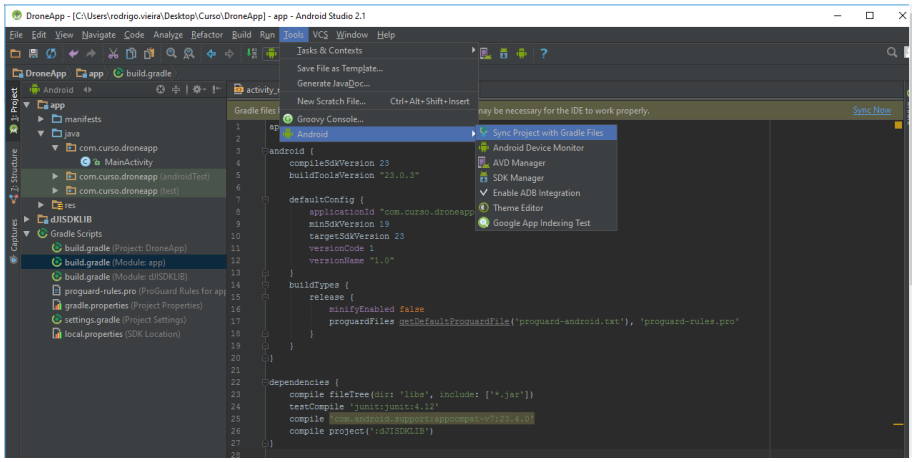


Figura 3.22. Sincronização dos arquivos Gradle com o projeto

Ao final da tarefa é possível verificar se o DJI Android SDK está totalmente configurado através da tela de configuração de módulos da aplicação. Clica-se com o botão direito em '*app*' no painel esquerdo do Android Studio e seleciona a opção '*Open Module Settings*'. Será aberto um popup, na aba '*Dependencies*' pode-se verificar se o módulo '*:djiSDKLIB*' está listado e em escopo de compilação, conforme apresentado na Figura 1.23.

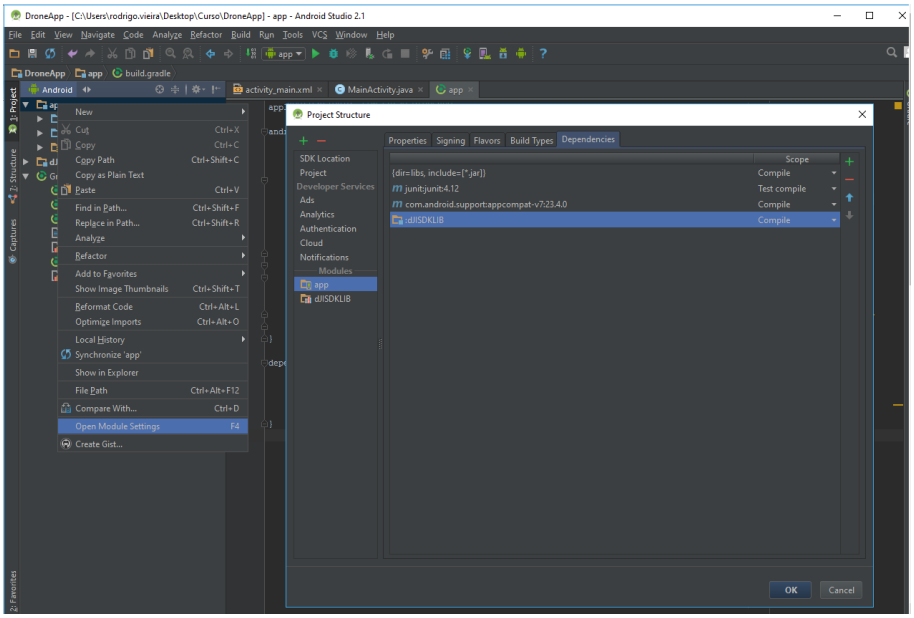


Figura 3.23. Verificação do status da importação

Neste momento aplicação Android está devidamente integrada com o SDK Android da DJI.

3.5.3. Registro da aplicação na página da DJI

Para desenvolver aplicações para drones da DJI é necessário criar uma conta de desenvolvedor na página da empresa e registrar a aplicação. No site https://developer.dji.com e selecionando o ícone no canto superior direito é possivel fazer o registro. O formulário requisita apenas informações de email e senha para criação da conta. Depois de criado o registro, pode-se fazer login [DJI Genegate App Key].

A tela principal de um usuário desenvolvedor da DJI é chamada *User Center*, conforme apresentado na Figura 1.24. Nesta tela pode-se registrar novas aplicações, visualizar e alterar as configurações das aplicações criadas e alterar suas informações de usuário. Para registrar uma aplicação basta clicar no botão 'CREATE APP'.

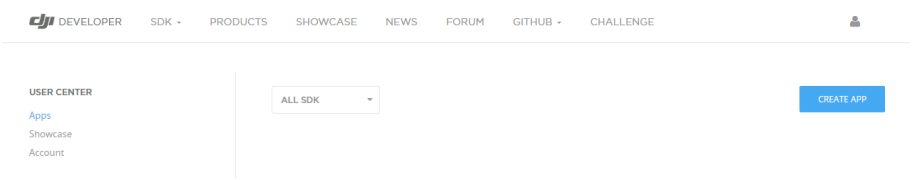


Figura 3.24. Tela de User Center da DJI

Um popup será aberto com o formulário de registro da sua aplicação. Para os fins deste cursooOs campos devem ser preenchidos conforme seguinte recomendação:

- Opção Mobile SDK no campo SDK;
- O nome da aplicação será DroneApp;
- A plataforma de desenvolvimento será Android;
- O nome do pacote deverá ser o mesmo nome do pacote da aplicação criada no Android Studio na seção anterior, no caso *droneapp.curso.com*;
- A categoria e descrição da aplicação é de livre preferência.

A figura abaixo mostra a configuração do app finalizada:

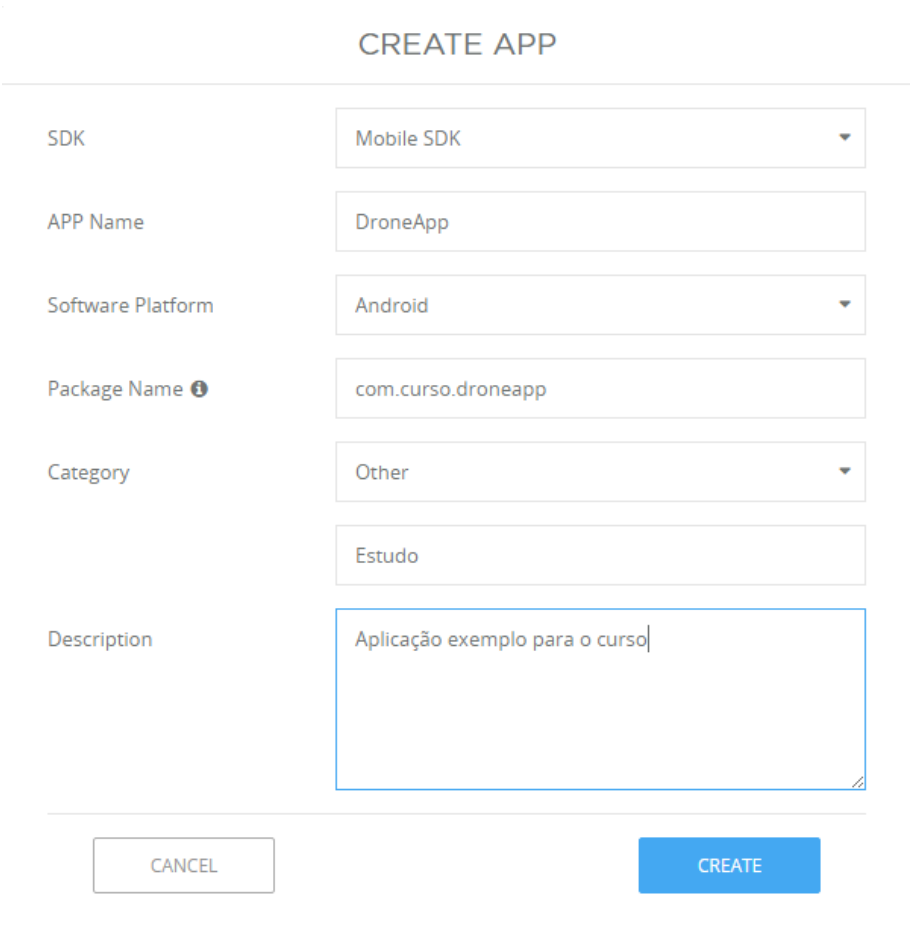


Figura 3.25. Formulário de registro da aplicação

A DJI vai enviar um email de ativação do aplicativo, para finalizar o registro da aplicação é preciso clicar neste link. Depois de finalizado o processo de registro da aplicação pode-se visualizar e editar as informações de registro conforme figura 1.26. A

App Key será utilizada para registro do SDK no momento da programação da aplicação, portanto é importante guardar esta informação.

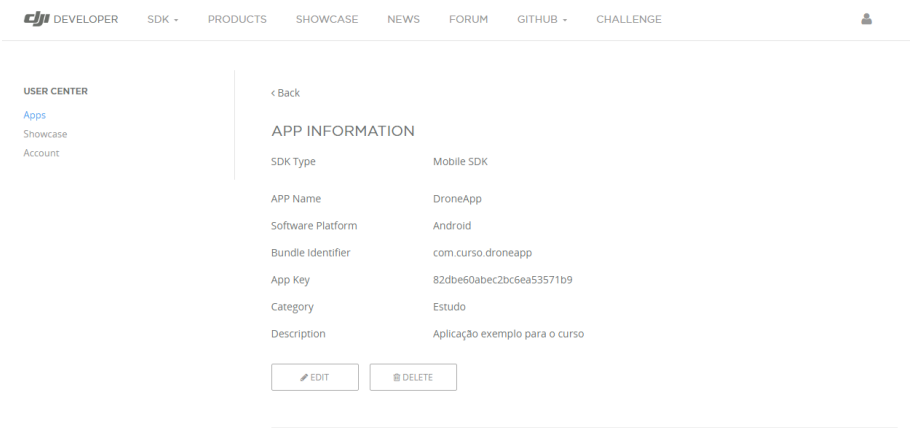


Figura 3.26. Dashboard da DJI com aplicação registrada

Pronto, a aplicação está registrada e pronta para ser desenvolvida utilizando o Mobile SDK da DJI.

3.5.4. Registrando aplicação no Android Studio

Para registrar a aplicação é preciso fazer os seguintes passos [DJI Workflow Integrate]:

Conceder à aplicação as permissões que o SDK da DJI necessita para registro da mesma:

```
1  <uses-permission android:name="android.permission.BLUETOOTH" />
2  <uses-permission android:name="android.permission.
      BLUETOOTH_ADMIN" />
3  <uses-permission android:name="android.permission.VIBRATE" />
4  <uses-permission android:name="android.permission.INTERNET" />
5  <uses-permission android:name="android.permission.
      ACCESS_WIFI_STATE" />
6  <uses-permission android:name="android.permission.WAKE_LOCK" />
7  <uses-permission android:name="android.permission.
      ACCESS_COARSE_LOCATION" />
8  <uses-permission android:name="android.permission.
      ACCESS_NETWORK_STATE" />
9  <uses-permission android:name="android.permission.
      ACCESS_FINE_LOCATION" />
10 <uses-permission android:name="android.permission.
      CHANGE_WIFI_STATE" />
11 <uses-permission android:name="android.permission.
      MOUNT_UNMOUNT_FILESYSTEMS" />
12 <uses-permission android:name="android.permission.
      WRITE_EXTERNAL_STORAGE" />
13 <uses-permission android:name="android.permission.
      READ_EXTERNAL_STORAGE" />
```

```
<uses-permission android:name="android.permission.
   SYSTEM_ALERT_WINDOW" />
<uses-permission android:name="android.permission.
   READ_PHONE_STATE" />
<uses-feature android:name="android.hardware.camera" />
<uses-feature android:name="android.hardware.camera.autofocus"
   />
<uses-feature
    android:name="android.hardware.usb.host"
    android:required="false" />
<uses-feature
    android:name="android.hardware.usb.accessory"
    android:required="true" />
```

Adicionalmente, a tag <uses-feature> é necessária para indicar que a aplicação vai precisar utilizar a câmera e um dispositivo USB, que é o controle do drone.

Dentro da tag <application> deve-se adicionar a App Key que é recebida no momento de registro da aplicação na página da DJI:

```
<!-- DJI SDK -->
<uses-library android:name="com.android.future.usb.accessory" />
<meta-data
    android:name="com.dji.sdk.API_KEY"
    android:value="82dbe60abec2bc6ea53571b9" /> // Please enter
       your APP Key here.
<activity
    android:name="dji.sdk.SDKManager.DJIAoaControllerActivity"
    android:theme="@android:style/Theme.Translucent" >
    <intent-filter>
        <action android:name="android.hardware.usb.action.
           USB_ACCESSORY_ATTACHED" />
    </intent-filter>
    <meta-data
        android:name="android.hardware.usb.action.
           USB_ACCESSORY_ATTACHED"
        android:resource="@xml/accessory_filter" />
</activity>
<service android:name="dji.sdk.SDKManager.DJIGlobalService" >
</service>
<!-- DJI SDK -->
```

3.5.5. Criando a classe DroneApplication para registro da aplicação

Neste exemplo de aplicação será criada uma classe chamada DroneApplication herdeira da classe Application do Android SDK, na qual será feito o registro da aplicação. Esta classe será utilizada, pois ela é instanciada antes de qualquer outra classe quando o processo da aplicação é criado. **??** A classe DroneApplication deve ser criada no pacote principal da aplicação - com.curso.droneapp - e deve extender a classe Application do SDK Android, vide código abaixo:

```
package com.curso.droneapp;

import android.app.Application;

public class DroneApplication extends Application {

}
```

O arquivo de configuração AndroidManifest.xml deve ser alterado para declarar a classe DroneApplication na tag <application>:

```
<application
  android:name=".DroneApplication"
  android:allowBackup="true"
  android:icon="@mipmap/ic_launcher"
  android:label="@string/app_name"
  android:supportsRtl="true"
  android:theme="@style/AppTheme">
```

Agora a classe DroneApplication deve implementar métodos para emitir uma mensagem caso a aplicação seja registrada com sucesso:

```
private static final String TAG = DroneApplication.class.getName
    ();
public static final String FLAG_CONNECTION_CHANGE = "
   dji_sdk_connection_change";
private static DJIBaseProduct mProduct;
private Handler mHandler;

@Override
public void onCreate() {
    super.onCreate();
    mHandler = new Handler(Looper.getMainLooper());
    DJISDKManager.getInstance().initSDKManager(this,
       mDJISDKManagerCallback);
}
```

O método initSDKManager tenta registrar a aplicação e recebe uma chamada de resposta na variável mDJISDKManagerCallback.

A variável mDJISDKManagerCallback implementa as funções onGetRegisteredResult e onProductChanged.

onGetRegisteredResult é o método de resposta após a tentativa de registro da aplicação, aqui o resultado é interpretado e uma mensagem é exibida para o usuário;

onProductChanged é o método que recebe o objeto DJIBaseProduct que significa que o Drone está devidamente conectado com a aplicação e caso a conexão com o Drone se perca, no momento da nova conexão a variável de produto deve ser atualizada sem que haja prejuízos.

```
private DJISDKManager.DJISDKManagerCallback
```

```
    mDJISDKManagerCallback = new DJISDKManager.
    DJISDKManagerCallback() {

  @Override
  public void onGetRegisteredResult(DJIError error) {
    Log.d(TAG, error == null ? "success" : error.getDescription
        ());
    if (error == DJISDKError.REGISTRATION_SUCCESS) {
        DJISDKManager.getInstance().startConnectionToProduct();
        Handler handler = new Handler(Looper.getMainLooper());
        handler.post(new Runnable() {
          @Override
          public void run() {
              Toast.makeText(getApplicationContext(), "Success",
                  Toast.LENGTH_LONG).show();
          }
        });
        Log.d(TAG, "Register success");

    } else {
        Handler handler = new Handler(Looper.getMainLooper());
        handler.post(new Runnable() {

          @Override
          public void run() {
              Toast.makeText(getApplicationContext(), "register
                 sdk fails, check network is available", Toast.
                 LENGTH_LONG).show();
          }
        });
        Log.d(TAG, "Register failed");
    }
    Log.e(TAG, error == null ? "success" : error.getDescription
        ());
  }

  @Override
  public void onProductChanged(DJIBaseProduct oldProduct,
     DJIBaseProduct newProduct) {

    mProduct = newProduct;
    if(mProduct != null) {
        mProduct.setDJIBaseProductListener(
           mDJIBaseProductListener);
    }
    notifyStatusChange();
  }
};

```

```
42 private DJIBaseProduct.DJIBaseProductListener
      mDJIBaseProductListener = new DJIBaseProduct.
      DJIBaseProductListener() {
43
44     @Override
45     public void onComponentChange(DJIBaseProduct.DJIComponentKey
           key, DJIBaseComponent oldComponent, DJIBaseComponent
          newComponent) {
46         if(newComponent != null) {
47             newComponent.setDJIComponentListener(
                  mDJIComponentListener);
48         }
49         notifyStatusChange();
50     }
51
52     @Override
53     public void onProductConnectivityChanged(boolean isConnected
          ) {
54         notifyStatusChange();
55     }
56
57 };
58
59 private DJIBaseComponent.DJIComponentListener
      mDJIComponentListener = new DJIBaseComponent.
      DJIComponentListener() {
60
61     @Override
62     public void onComponentConnectivityChanged(boolean
          isConnected) {
63         notifyStatusChange();
64     }
65
66 };
67
68 private void notifyStatusChange() {
69     mHandler.removeCallbacks(updateRunnable);
70     mHandler.postDelayed(updateRunnable, 500);
71 }
72
73 private Runnable updateRunnable = new Runnable() {
74
75     @Override
76     public void run() {
77         Intent intent = new Intent(FLAG_CONNECTION_CHANGE);
78         sendBroadcast(intent);
79     }
80 };
```

Nos dispositivos Android com versão superior ou igual à Marshmallow é preciso fazer o tratamento de permissões em tempo de execução, para isso a classe MainActivity precisa implementar o método *requestPermissions* da seguinte forma:

```
1  public class MainActivity extends AppCompatActivity {
2
3    @Override
4    protected void onCreate(Bundle savedInstanceState) {
5      super.onCreate(savedInstanceState);
6      setContentView(R.layout.activity_main);
7
8      if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
9          ActivityCompat.requestPermissions(this,
10           new String[]{Manifest.permission.
                WRITE_EXTERNAL_STORAGE, Manifest.permission.VIBRATE
                ,
11                   Manifest.permission.INTERNET, Manifest.
                        permission.ACCESS_WIFI_STATE,
12                   Manifest.permission.WAKE_LOCK, Manifest.
                        permission.ACCESS_COARSE_LOCATION,
13                   Manifest.permission.ACCESS_NETWORK_STATE,
                        Manifest.permission.ACCESS_FINE_LOCATION,
14                   Manifest.permission.CHANGE_WIFI_STATE,
                        Manifest.permission.
                        MOUNT_UNMOUNT_FILESYSTEMS,
15                   Manifest.permission.READ_EXTERNAL_STORAGE,
                        Manifest.permission.SYSTEM_ALERT_WINDOW,
16                   Manifest.permission.READ_PHONE_STATE,
17           }
18           , 1);
19     }
20   }
21 }
```

Toda vez que a aplicação é aberta, o método onCreate da classe DroneApplication será executado e uma mensagem será exibida se a aplicação foi registrada com sucesso, conforme apresentado na Figura 1.27

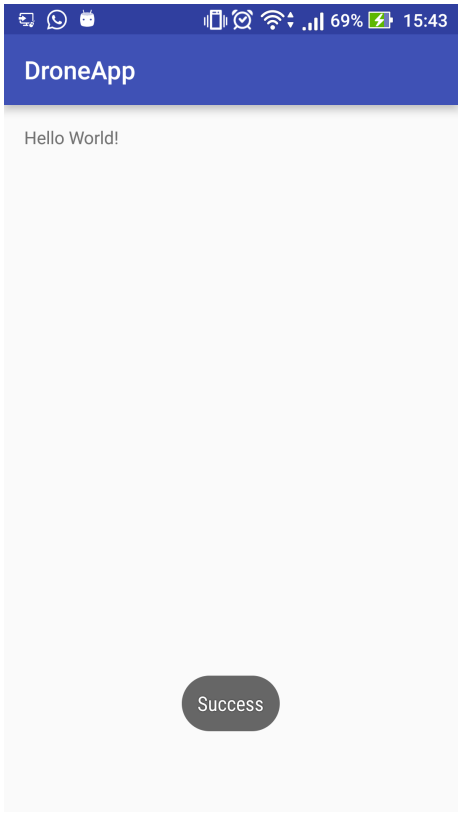


Figura 3.27. Aplicação registrada com sucesso

Agora a aplicação está devidamente registrada.

3.6. Criando aplicação de câmera ao vivo

As informações utilizadas nessa seção foram obtidas e adaptadas do guia oficial da DJI para construção de uma aplicação de câmera [DJI Camera App Tutorial], adicionalmente as informações referentes à classes e objetos do Mobile SDK da DJI foram encontradas na página de documentação do SDK [DJI Android SDK Reference].

Para criar uma aplicação de transmissão da câmera do Drone ao vivo para dispositivos Android, é será criada uma nova tela, que será chamada CameraModeActivity .

Antes de desenvolver a nova tela, é preciso conectar a MainActivity com a CameraModeActivity, para isso será adicionado um botão no arquivo de interface, activity_main.xml,este botão nos dará acesso à CameraModeActivity:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <RelativeLayout xmlns:android="http://schemas.android.com/apk/
     res/android"
3   xmlns:tools="http://schemas.android.com/tools"
4   android:layout_width="match_parent"
5   android:layout_height="match_parent"
6   android:paddingBottom="@dimen/activity_vertical_margin"
7   android:paddingLeft="@dimen/activity_horizontal_margin"
```

```
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context="com.curso.droneapp.MainActivity">

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Modo Camera"
    android:id="@+id/cameraModeBTN"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true"/>

</RelativeLayout>
```

A classe MainActivity deve inicializar a variável de botão e tratar o evento de clique, no qual será aberta a nova tela CameraModeActivity, vide código abaixo:

```
public class MainActivity extends AppCompatActivity implements
   View.OnClickListener {

  Button cameraModeBTN;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        ActivityCompat.requestPermissions(this,
          new String[]{Manifest.permission.
             WRITE_EXTERNAL_STORAGE, Manifest.permission.VIBRATE
             ,
                  Manifest.permission.INTERNET, Manifest.
                     permission.ACCESS_WIFI_STATE,
                  Manifest.permission.WAKE_LOCK, Manifest.
                     permission.ACCESS_COARSE_LOCATION,
                  Manifest.permission.ACCESS_NETWORK_STATE,
                     Manifest.permission.ACCESS_FINE_LOCATION,
                  Manifest.permission.CHANGE_WIFI_STATE,
                     Manifest.permission.
                     MOUNT_UNMOUNT_FILESYSTEMS,
                  Manifest.permission.READ_EXTERNAL_STORAGE,
                     Manifest.permission.SYSTEM_ALERT_WINDOW,
                  Manifest.permission.READ_PHONE_STATE,
          }
          , 1);
    }

    initUI();
  }
```

```
25
26    private void initUI() {
27      cameraModeBTN = (Button) findViewById(R.id.cameraModeBTN);
28      cameraModeBTN.setOnClickListener(this);
29    }
30
31    @Override
32    public void onClick(View v) {
33      if (v.getId() == R.id.cameraModeBTN) {
34          startCameraModeActivity();
35      }
36    }
37
38    private void startCameraModeActivity() {
39      startActivity(new Intent(this, CameraModeActivity.class));
40    }
41  }
```

Conexão feita, é preciso criar a classe CameraModeActivity. Vide seção 1.4.1.2. A tela de câmera terá uma TextureView que ocupará toda a tela do dispositovo e um botão para captura de photos. O arquivo de configuração de interface, activity_camera_mode.xml, ficará da seguinte forma:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout xmlns:android="http://schemas.android.com/apk/
       res/android"
3    xmlns:tools="http://schemas.android.com/tools"
4    android:layout_width="match_parent"
5    android:layout_height="match_parent"
6    tools:context="com.curso.droneapp.MainActivity">
7
8    <TextureView
9        android:layout_width="match_parent"
10       android:layout_height="match_parent"
11       android:id="@+id/cameraScreenTV"/>
12
13   <ImageButton
14       android:layout_width="wrap_content"
15       android:layout_height="wrap_content"
16       android:id="@+id/capturePhotoIMB"
17       android:src="@android:drawable/ic_menu_camera"
18       android:layout_alignParentEnd="true"
19       android:layout_centerVertical="true"
20       android:layout_marginEnd="20dp"/>
21
22 </RelativeLayout>
```

Como a aplicação precisará de toda a tela do dispositivo e a imagem do Drone é em modo paiságem, devemos declarar no AndroidManifest como segue:

```
<activity android:name=".CameraModeActivity"
    android:screenOrientation="landscape"
    android:theme="@android:style/Theme.NoTitleBar">
</activity>
```

Na classe CameraModeActivity serão necessárias as seguintes variáveis:

- mReceivedVideoDataCallBack é a variável responsável por obter os dados da câmera do Drone e decodificar a informação para o dispositivo Android;
- mCodecManager é utilizado pela mReceivedVideoDataCallBack como o decoder da transmissão de vídeo do Drone;
- cameraScreenTV é o TextureView criado no arquivo activity_camera_mode.xml para exibir o vídeo;
- capturePhotoIMB é o botão para captura de foto;
- O método setResultToToast é utilizado para exibir uma mensagem de resposta ao usuário;
- mReceiver é um objeto que fica 'escutando' a conexão da aplicação com o Drone e possibilita o tratamento para o caso de alterações.

O resultado é o seguinte:

```
protected static final String TAG = "CameraModeActivity";
protected DJICamera.CameraReceivedVideoDataCallback
   mReceivedVideoDataCallBack;
protected DJICodecManager mCodecManager;

ImageButton capturePhotoIMB;
TextureView cameraScreenTV;

@Override
protected void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_camera_mode);
    initUI();
}

private void initUI() {
    cameraScreenTV = (TextureView) findViewById(R.id.
       cameraScreenTV);
    capturePhotoIMB = (ImageButton) findViewById(R.id.
       capturePhotoIMB);
    capturePhotoIMB.setOnClickListener(this);

    mReceivedVideoDataCallBack = new DJICamera.
       CameraReceivedVideoDataCallback() {
```

```
21
22          @Override
23          public void onResult(byte[] videoBuffer, int size) {
24              if (mCodecManager != null) {
25                  // Send the raw H264 video data to codec manager
                        for decoding
26                  mCodecManager.sendDataToDecoder(videoBuffer,
                        size);
27              } else {
28                  Log.e(TAG, "mCodecManager is null");
29              }
30          }
31      };
32
33      IntentFilter filter = new IntentFilter();
34      filter.addAction(DroneApplication.FLAG_CONNECTION_CHANGE);
35      registerReceiver(mReceiver, filter);
36  }
37
38  private void setResultToToast(String message) {
39      Toast.makeText(this, message, Toast.LENGTH_SHORT).show();
40  }
41
42  protected BroadcastReceiver mReceiver = new BroadcastReceiver()
        {
43      @Override
44      public void onReceive(Context context, Intent intent) {
45          onProductChange();
46      }
47  };
48
49  protected void onProductChange() {
50      initPreviewer();
51  }
```

Agora que as variáveis de interface de comunicação com o Drone está inicializadas, é possível iniciar a reprodução do vídeo recebido da câmera do Drone:

- o método initPreviewer obtém uma instância do objeto de controle da câmera do Drone;
- o mReceivedVideoDataCallBack é registrado como objeto de descriptação do vídeo da câmera;
- o TextureView cameraScreenTV é registrado para receber o vídeo e tratá-lo através dos métodos onSurfaceTextureAvailable, onSurfaceTextureSizeChanged, onSurfaceTextureDestroyed e onSurfaceTextureUpdated, que são métodos da interface TextureView.SurfaceTextureListener;

- o método uninitPreviewer é utilizado para tratar quando a aplicação é fechada ou colocada em background, neste caso precisamos desregistrar o mReceivedVideo-DataCallBack.

```
1  private void initPreviewer() {
2
3    DJIBaseProduct product = DroneApplication.getProductInstance()
       ;
4
5    if (product == null || !product.isConnected()) {
6        setResultToToast("Aircraft is disconnected");
7    } else {
8        if (null != cameraScreenTV) {
9            cameraScreenTV.setSurfaceTextureListener(this);
10       }
11       if (!product.getModel().equals(DJIBaseProduct.Model.
           UnknownAircraft)) {
12           DJICamera camera = product.getCamera();
13           if (camera != null){
14               // Set the callback
15               camera.setDJICameraReceivedVideoDataCallback(
                     mReceivedVideoDataCallBack);
16           }
17       }
18   }
19 }
20
21 private void uninitPreviewer() {
22   DJICamera camera = DroneApplication.getCameraInstance();
23   if (camera != null){
24       // Reset the callback
25       DroneApplication.getCameraInstance().
             setDJICameraReceivedVideoDataCallback(null);
26   }
27 }
28
29 @Override
30 public void onSurfaceTextureAvailable(SurfaceTexture surface,
       int width, int height) {
31   Log.e(TAG, "onSurfaceTextureAvailable");
32   if (mCodecManager == null) {
33       mCodecManager = new DJICodecManager(this, surface, width,
             height);
34   }
35 }
36
37 @Override
38 public void onSurfaceTextureSizeChanged(SurfaceTexture surface,
       int width, int height) {
```

```
  Log.e(TAG, "onSurfaceTextureSizeChanged");
}

@Override
public boolean onSurfaceTextureDestroyed(SurfaceTexture surface)
    {
  Log.e(TAG,"onSurfaceTextureDestroyed");
  if (mCodecManager != null) {
      mCodecManager.cleanSurface();
      mCodecManager = null;
  }
  return false;
}

@Override
public void onSurfaceTextureUpdated(SurfaceTexture surface) {
}
```

Para tratar o ciclo de vida da tela devemos implementar os métodos onResume, onPause e onDestroy como segue:

```
@Override
protected void onResume(){
    super.onResume();
    initPreviewer();
}

@Override
protected void onPause() {
    uninitPreviewer();
    super.onPause();
}

@Override
protected void onDestroy(){
    super.onDestroy();
    uninitPreviewer();
    unregisterReceiver(mReceiver);
}
```

Agora que a transmissão da câmera já está devidamente configurada, podemos implementar a captura de fotos:

- a variável *camera* obtém a instância *DJICamera* que representa a câmera do Drone;
- é preciso configurar a câmera para o modo de captura de foto, para isso a variável *photoMode* é a constante *DJICameraSettingsDef.CameraShootPhotoMode.Single*;
- a variável *camera* possui um método chamado *startShootPhoto* com o qual consegue-se a captura de fotos.

O código abaixo representa o resultado esperado:

```
@Override
public void onClick(View v) {
    if (v.getId() == R.id.capturePhotoIMB) {
        capturePhoto();
    }
}

private void capturePhoto() {
  final DJICamera camera = DroneApplication.getCameraInstance();
  if (camera != null) {
      DJICameraSettingsDef.CameraShootPhotoMode photoMode =
         DJICameraSettingsDef.CameraShootPhotoMode.Single; //
         Set the camera capture mode as Single mode
      camera.startShootPhoto(photoMode, new DJIBaseComponent.
         DJICompletionCallback() {

          @Override
          public void onResult(DJIError error) {
              if (error == null) {
                  setResultToToast("Take photo: success");
              } else {
                  setResultToToast(error.getDescription());
              }
          }

      });
  } else {
      setResultToToast("Aircraft not connected!");
  }
}
```

Está pronta a aplicação que recebe o vídeo da câmera do drone em tempo real e pode capturar fotos que serão salvas no cartão SD do Drone.

3.7. Obtendo informação de Contexto a partir de Sensores

O Kit de desenvolvimento da DJI para aplicativos móveis, DJI MobileSDK, oferece uma série de funcionalidades de comunicação com o Drone que permitem que uma aplicação móvel possa obter todas as informações sensoriais do Drone. Nessa seção vamos abordar alguns desses sensores e ensinar como podemor desenvolver aplicações que fazem uso dos mesmos.

3.7.1. Geolocalização

Obter a localização do seu Drone é uma tarefa bastante simples, o DJI Mobile SDK fornece uma classe chamada DJIFlightController que possibilita enviar comandos de voo e receber informações de sensores do Drone. Dentre os objetos que compõe o DJIFlightController, o que nos interessa é o DJIFlightControllerCurrentState que agrega as infor-

mações de status do Drone, como localização, status dos motores, tempo de voo, velocidade, entre outros. Obter as coordenadas do Drone pode ser implementado da seguinte forma:

```
DJIFlightController mFlightController = null;

// Precisamos do objeto DJIBaseProduct para obter o objeto DJIFlightController
DJIBaseProduct product = FPDroneApplication.getProductInstance();
if (product != null && product.isConnected()) {
    if (product instanceof DJIAircraft) {
        mFlightController = ((DJIAircraft) product).getFlightController();
    }
}

// Registramos um objeto para ficar 'escutando' as alteracoes de status do Drone atraves do metodo setUpdateSystemStateCallback
// o metodo onResult e chamado toda vez que o status do Drone e alterado
if (mFlightController != null) {
    mFlightController.setUpdateSystemStateCallback(new DJIFlightControllerDelegate.FlightControllerUpdateSystemStateCallback() {

        @Override
        public void onResult(DJIFlightControllerDataType.DJIFlightControllerCurrentState state) {
            droneLocationLat = state.getAircraftLocation().getLatitude();
            droneLocationLng = state.getAircraftLocation().getLongitude();
            droneAltitude = state.getAircraftLocation().getAltitude();
        }
    });
}
```

Cruzando as coordenadas de localização recebidas do Drone com uma aplicação de Mapas, exemplo Google Maps, podemos contextualizar a informação que antes era apenas um número e dar um sentido à nossa aplicação. Exemplo, uma aplicação que faça o Drone parar quando estiver acima de uma estação de metrô e tire fotos do local.

3.7.2. Bússola

O sensor de bússula indica para qual direção o Drone está apontando e pode ser utilizado, por exemplo, para aplicações de automação de rotas. A classe DJIFlightController contém uma classa chamada DJICompass que possui métodos para calibração da bússola e

obtenção de para onde o Drone está apontando. Esta informação pode ser acessada de forma semelhante à mencionada na subseção 1.7.1:

```
mFlightController.setUpdateSystemStateCallback(new
    DJIFlightControllerDelegate.
    FlightControllerUpdateSystemStateCallback() {

     @Override
     public void onResult(DJIFlightControllerDataType.
         DJIFlightControllerCurrentState state) {
          finalMFlightController.getCompass().getHeading();
     }
});
```

Poderíamos criar uma aplicação que planejasse rotas de acordo com a orientação da bússula, por exemplo, 50 metros ao norte, 35 metros a nordeste, etc.

3.7.3. Sensores baseados na câmera

A DJI oferece um kit de desenvolvimento específico para aplicações baseadas em reconhecimento via câmera, é o GuidanceSDK. Esse SDK oferece funcionalidades de reconhecimento e mapeamento de terreno com as quais podemos criar aplicações de varredura de terreno, busca de melhor caminho entre obstáculos, reconhecimento de objetos ao redor do Drone, etc.

3.8. Conclusões

A computação ubíqua é um termo utilizado para expressar a informática de forma onipresente no cotidiano. A ideia da computação ubíqua é que a computação será onipresente, estendendo os limites da computação para fora dos computadores, tornando-se pervasiva no cotiano, de forma “invisível”. Drones são uma poderosa plataforma de desenvolvimento de aplicações úbiquas visto que eles consistem, de maneira genérica, em um computador móvel, repleto de sensores (câmera, gps, sensores de pressão, sensores de velocidade, acelelômetros, dentre outros) e livres de restrições de movimentos (drones voam) que podem ser utizados como uma extensão física do usuário pois, possibilitam uma integração entre o usuário e meio de uma forma não anteriomente permitida, abrindo espaço para obtenção de uma enorme quantidade de dados, o que abre uma gama de aplicações possíveis.

Neste capítulo mostramos como configurar o ambiente de desenvolvimento e criar uma aplicação móvel para controlar um drone Phantom 4 através de um dispositivo android. Também mostramos como é possível obter os dados dos sensores do drone através das funcionalidades do Mobile SDK da DJI. Estas são as atividades básicas para o desenvolvimento de aplicações móveis com a utilização de drones. A partir deste ponto, a documentação do Mobile SDK da DJI é suficiente para a construção de qualquer aplicação que necessite de comunicação entre um dispositivo móvel e um Drone.

Referências

[dev] Android sdk - application. https://developer.android.com/studio/intro/index.html. Accessado: 16/09/2016.

[sbc] Grandes desafios da pesquisa em computação no brasil – 2006 – 2016. http://www.gta.ufrj.br/rebu/arquivos/SBC-Grandes.pdf. Accessado: 16/09/2016.

[nsf] Nsf - national science foundation. http://www.nsf.gov. Accessado: 16/09/2016.

[Kaufmann and Buechley 2010] Kaufmann, B. and Buechley, L. (2010). Amarino: a toolkit for the rapid prototyping of mobile ubiquitous computing. In *Proceedings of the 12th international conference on Human computer interaction with mobile devices and services*, pages 291–298. ACM.

[Krumm 2016] Krumm, J. (2016). *Ubiquitous computing fundamentals*. CRC Press.

[Google Android Studio Guide] Google Android Studio Guide. https://developer.android.com/studio/intro/index.html. Accessado: 16/09/2016.

[DJI Workflow Integrate] DJI Workflow Integrate. https://developer.dji.com/mobile-sdk/documentation/application-development-workflow/workflow-integrate.html#android-studio-project-integration. Accessado: 16/09/2016.

[DJI Genegate App Key] DJI Genegate App Key. https://developer.dji.com/mobile-sdk/documentation/quick-start/index.html#generate-an-app-key. Accessado: 16/09/2016.

[DJI Onboard SDK] DJI Onboard SDK. https://developer.dji.com/onboard-sdk/. Accessado: 16/09/2016.

[DJI Mobile SDK] DJI Mobile SDK. https://developer.dji.com/mobile-sdk/. Accessado: 16/09/2016.

[DJI Guidance SDK] DJI Guidance SDK. https://developer.dji.com/guidance-sdk/. Accessado: 16/09/2016.

[DJI Camera App Tutorial] DJI Camera Application Tutorial. https://developer.dji.com/mobile-sdk/documentation/android-tutorials/FPVDemo.html. Accessado: 16/09/2016.

[DJI Android SDK Reference] DJI Android SDK API Reference. https://developer.dji.com/iframe/mobile-sdk-doc/android/reference/packages.html. Accessado: 16/09/2016.

Capítulo 4

Processamento Digital de Imagens Médicas usando a Biblioteca Livre ITK

Alexandre Ribeiro Cajazeira Ramos, Antonio Oseas de Carvalho Filho, Marcos Raniere de Sousa Silva

Abstract

From the problems generated by several diseases worldwide, techniques that support the work of health professionals have been studied in various fields of knowledge. In computing, one can highlight the expert aid systems, which provide additional information of medical exams and are able to contribute significantly to interpretations and decisions. This short course aims to introduce the public library Insight Segmentation and Registration Toolkit-ITK, comprising a set of techniques and tools for manipulation and analysis of medical images, with open source and multiplatform. By ITK can scrutinize and understand specific techniques of medical image handling and build complete systems of specialist aid.

Resumo

A partir da problemática gerada por diversas doenças em nível mundial, técnicas que auxiliem o trabalho dos profissionais de saúde vêm sendo estudadas nas mais diversas áreas do conhecimento. Na computação, pode-se destacar os sistemas de auxílio ao especialista, que fornecem informações complementares de exames médicos e são capazes de contribuir significativamente às interpretações e tomadas de decisão de alto risco. Este minicurso objetiva apresentar a biblioteca pública Insight Segmentation and Registration Toolkit - ITK, que compreende um conjunto de técnicas e ferramentas de manipulação e análise de imagens médicas, com código aberto e multiplataforma. Através do ITK pode-se esmiuçar e compreender técnicas específicas de manipulação de imagens médicas e construir sistemas completos de auxílio ao especialista.

4.1. Introdução

4.1.1. Processamento Digital de Imagens

Entende-se por Processamento Digital Imagens (PDI) como a manipulação de uma imagem por computador, de modo que a entrada e a saída desse processo é uma

imagem. O objetivo de se usar processamento digital de imagens é melhorar o aspecto visual de certas feições estruturais para o analista humano e fornecer outros subsídios para a sua interpretação, inclusive gerando produtos que possam ser posteriormente submetidos a outros processamentos [SPRING 1996].

De acordo com [Silva 2001], a principal função do processamento digital de imagens é fornecer ferramentas para facilitar a identificação e a extração de informações contidas nas imagens, para posterior interpretação. Nesse sentido, sistemas computacionais são utilizados para atividades interativas de análise e manipulação das imagens. O resultado desse processo é a produção de outras imagens, estas já contendo informações específicas, extraídas e realçadas a partir das imagens originais.

O processamento da imagem não é uma tarefa simples, pois é feito em várias etapas interconectadas. Algumas etapas do processamento digital de imagens são Aquisição, Pré-processamento, Segmentação, Extração de Características e, Reconhecimento e Interpretação. A figura 4.1 ilustra um diagrama para um sistema de PDI.

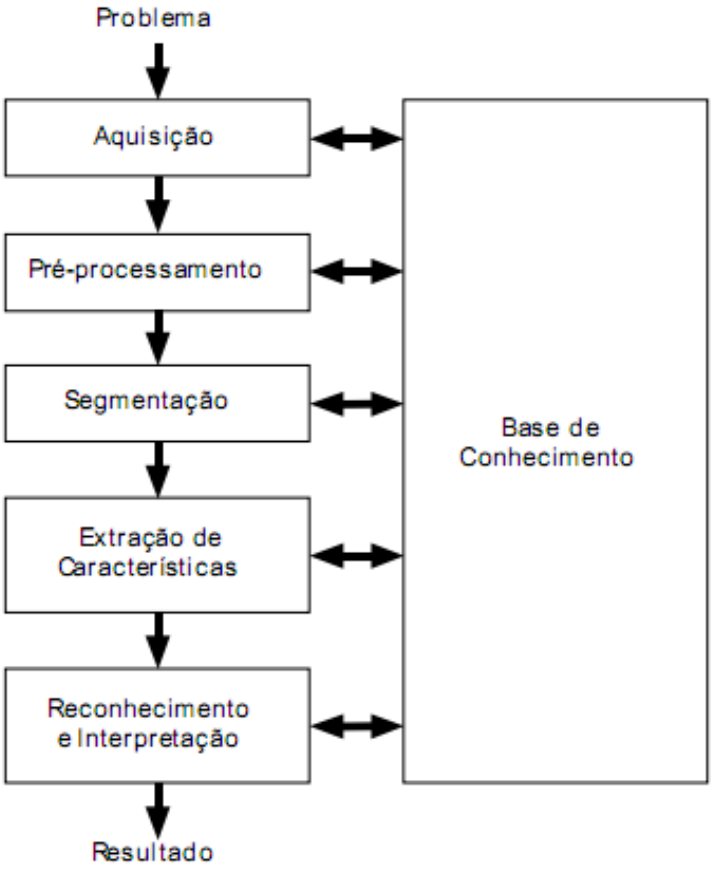


Figura 4.1. Etapas de um Sistema PDI. Fonte: [Filho e Neto 1999]

Na primeira etapa, tem-se a aquisição de imagens. É nessa etapa que as imagens são capturadas e representadas de forma computacional para serem interpretadas na etapa seguinte.

No pré-processamento, são realizados procedimentos capazes de proporcionar um melhoramento nos aspectos visuais e estruturais da imagem. Dessa maneira, consegue-se aumentar contraste, eliminar ruídos, etc.

A etapa de segmentação consiste em dividir a imagem em objeto (s) e fundo. Em outras palavras, essa etapa consiste em técnicas que de alguma maneira consigam formar padrões de agrupamento, gerando sub-regiões que possuem entre si alguma similaridade.

A representação e descrição, conhecida também por extração de características, tem como objetivo representar, através de valores, uma imagem ou partes dela. Estes valores são características fundamentais que representam propriedades contidas nas imagens.

Entende-se por base de conhecimento, toda e qualquer informação útil para o processamento como um todo, sendo esta composta em geral por imagens, informações de especialistas, dentre outros. Essas informações são úteis pois serão usadas em todas as etapas, com destaque para a etapa do reconhecimento e interpretação.

E por fim, tem-se a etapa de reconhecimento de padrões. Nessa etapa, os valores obtidos na etapa de extração de características são os insumos para que uma técnica de aprendizado de máquina possa, então, discernir entre possíveis padrões contidos em um grupo de imagens.

4.1.2. Informática Médica

A partir da problemática gerada por diversas doenças em nível mundial, uma gama de técnicas que auxiliam o trabalho dos profissionais de saúde tem sido estudadas e implementadas nas mais diversas áreas do conhecimento. Na computação, podemos pontuar os sistemas de auxílio ao especialista, que fornecem informações complementares de exames médicos e são capazes de contribuir significativamente às interpretações e tomadas de decisão de alto risco. Para construção destes sistemas são associadas técnicas relacionadas a Processamento digital de imagens, Inteligência artificial, Aprendizado de máquina, dentre outras áreas de pesquisa, apresentando respostas significativas às demandas apresentadas.

A Informática médica compreende um conjunto de soluções computacionais capazes de auxiliar no trabalho dos especialistas de saúde e na relação dos pacientes com diversas doenças, como demonstrado nos exemplos apresentados acima. Esta área de conhecimento perpassa desde os sistemas de auxílio ao diagnóstico a aplicativos que realizam a prevenção ou acompanhamento de diversas doenças. Dessa maneira, tendo em vista a relevância da saúde na vida das pessoas e sociedade, todos os esforços neste sentido são significativos e satisfatórios.

Os sistemas especialistas fornecem informações complementares capazes de enriquecer e garantir a consistência das interpretações dos especialistas. Estes sistemas são divididos entre os sistemas de auxílio à detecção de lesões e de auxílio ao diagnóstico de lesões, ambos capazes de analisar informações sobre qualquer parte do corpo humano.

Através destes sistemas podemos observar uma contribuição significativa ao trabalho do especialista e, consequentemente, nos resultados para os pacientes. Um exemplo é a metodologia proposta por [Carvalho Filho et al. 2016], para a classificação de nódulos pulmonares pertencentes à base de imagem *LIDC-IDRI*. Foram utilizados os índices de diversidade taxonômica e a distinção taxonômica da ecologia para descrever a textura de nódulos e não-nódulos e para classificação foi utilizado a Máquina de etor

de Suporte - MVS. A metodologia foi empregada em 833 exames e obtve uma precisão média de 98,11% em sua classificação.

[Sampaio 2015] propôs uma metodologia para o diagnóstico automático de câncer de mama utilizando técnicas de melhoramento, segmentação, extração de característica de textura e forma, seleção de características e classificação. Para tanto, ele explorou as técnicas de algoritmos genéticos, máquina de vetor de suporte, *Density-Based Spatial Clustering of Applications with Noise* (DBSCAN), dentre outras. Os melhores resultados obtidos produziram uma sensibilidade de 94,02%, especificidade de 82,28% e acurácia de 84,08%, com uma taxa de 0,85 falsos positivos por imagem com uma área sob a *Free-Response ROC Curve* (curva FROC) de 1,13 nas análises de mamas não densas. Para mamas densas, obteve-se uma sensibilidade de 89,13%, especificidade de 88,61% e acurácia de 88,69%, com uma taxa de 0,71 falsos positivos por imagem com uma área sob a curva ROC de 1,47.

[CLARO *et al*. 2015] desenvolveu um método para detecção automática de Glaucoma, que é a segunda principal causa de cegueira no mundo e não possui cura. A metodologia utilizada no estudo foi: aquisição de imagem, pré-processamento nas imagens da retina, extração de características de cor e entropia na área alvo e logo após a seleção de atributos. Os melhores resultados produziram uma sensibilidade de 93,7%, especificidade de 93,6%, acurácia de 93,67%, F-*measure* de 93.6 e 0.83 no índice *Kappa*.

Os exemplos de aplicações da informática médica apresentadas utilizaram, dentre os recursos, a biblioteca pública *Insight Segmentation and Registration Toolkit* - ITK, que compreende um conjunto de técnicas e ferramentas de manipulação e análise de imagens médicas. O ITK pode contribuir na construção de sistemas completos de auxílio ao especialista, esmiuçar e compreender técnicas específicas de manipulação de imagens, ou construir novas abordagens e metodologias que contribuam para o contexto da informática médica.

4.2. *Insight Segmentation and Registration Toolkit – ITK*

4.2.1. Introdução ao ITK

A biblioteca livre *Insight Segmentation and Registration Toolkit* (ITK) é um sistema multi-plataforma de código aberto que fornece aos desenvolvedores um extenso conjunto de ferramentas de software para processamento de imagens. Desenvolvido através de metodologias ágeis, o ITK emprega algoritmos de ponta para registro e segmentação de dados multidimensionais [ITK 2016].

O ITK foi criado pela *National Library of Medicine* (NLM) com a seguintes metas:

- Apoiar o projeto humano visível.
- Estabelecer uma base para futuras pesquisas.
- Criar um repositório de algoritmos fundamentais.
- O desenvolvimento de uma plataforma para o desenvolvimento de produtos avançados.
- Apoiar a aplicação comercial da tecnologia.

- Criar convenções para o trabalho futuro.
- Crescer uma comunidade autossustentável de usuários e desenvolvedores de software.

O ITK possui algoritmos e estruturas para executar segmentação e registro de imagens. Segmentação é definido como o processo de identificar e classificar os elementos encontrados em uma imagem digital. Já o registro é a tarefa de alinhar a imagem, por meio de algum tipo de critério de correspondência entre dados. Essa biblioteca foca em imagens médicas, embora não haja restrições quanto ao processamento de outros tipos de dados. O ITK não realiza visualização de imagens, deixando a cargo de outras bibliotecas, como o VTK.

A biblioteca ITK é multi-plataforma implementada na linguagem C++, mas possui uma interface que permite a utilização através de outras linguagens como Java e Python. Por possuir programação genérica, o ITK apresenta flexibilidade em suas aplicações, muitos templates em C++ já são disponíveis.

Outra característica importante do ITK é ser um software livre, desta forma, ele é continuamente desenvolvido por colaboradores de todo o mundo, através do modelo de desenvolvimento *extreme programming*, o que consiste em um rápido, dinâmico e constante ciclo de projeto, programação, teste e lançamento. As atualizações são realizadas diariamente, assim essa biblioteca se mantém em constante evolução.

4.2.2. Arquitetura

O ITK possui implementação baseada em programação genérica, assim, há bastante uso de templetes, que ficam disponíveis aos usuários. É também um sistema mult-plataforma, utilizando o ambiente *CMaker* para configurar o processo de compilação em diferentes plataformas.

A arquitetura do ITK é baseada em fluxo de dados, pois os dados são representados por objetos de dados, que são processados por objetos de processamento conhecidos como filtros. Os objetos de dados e de processamento são conectados formando fluxos de processamento (pipeline). A arquitetura também possui, fabricas de objetos que são utilizadas para instanciar objetos [LOUREÇO 2011].

Os conceitos básicos da programação em ITK são:

1. Objeto de dados é a entidade que representa os dados em si e fornece acesso a eles. Um objeto de dados pode ser representado pelos tipos *itk::Image* ou *itk::Mesh*.
2. O filtro ou objeto de processamento, que é uma entidade que recebe um conjunto de uma ou mais entradas de dados, realiza operações com elas e retorna um conjunto de uma ou mais saídas de dados.
3. O fluxo de processamento de dados, ou pipeline, que é um grafo dirigido de processos e objetos de dados. O fluxo recebe, opera e retorna um objeto de dados.
4. Uma região de malha, ou mesh, que representa uma região não estruturada.
5. Uma região que representa uma parte estruturada dos dados.

Os objetos de processamento ou filtros são definidos de diversas formas, variando na quantidade e tipos de entradas e saídas. Alguns tipos de filtros são: *ImageToImageFilter* que recebe uma imagem e produz uma nova imagem. *UnaryFunctorImageFilter* é usado para definir um filtro que aplica uma função sobre uma imagem, entre outros. O processo de execução de um fluxo processamento é realizado seguindo os passos abaixo:

1. Determinar quais filtros precisam ser executados.
2. Inicializar os objetos de saída de dados preparando-os para novos dados.
3. Determinar a quantidade de dados que cada filtro deve processar para produzir uma saída de tamanho suficiente para os filtros subsequentes, levando em conta os limites de memória ou requisitos específicos de cada filtro.
4. Dividir os dados em pedaços de tamanho definido no passo anterior. Essa operação é chamada de streaming é e utilizada para o processamento em *multithreading* nativo do ITK e para o processamento de imagens que não cabem em memória.
5. Liberar a saída de dados dos filtros que não forem mais necessárias.

4.2.3. Aplicações

As aplicações construídas a partir do ITK perpassam, principalmente, pelos diversos processos de manipulação de imagens médicas, mas não se atêm, necessariamente, a este segmento do processamento digital de imagens. Suas ferramentas, filtros e funcionalidades permitem manipular toda e qualquer imagem digital, extraindo informações pertinentes, realçando determinadas características, realizando alterações, dentre outras operações.

As aplicações com ITK estão presentes na academia e indústria. Os primeiros contratos com a biblioteca envolveram: *Electric Corporate R&D, Kitware*, Universidade da Carolina do Norte, Universidade da Pennsylvania e Universidade de Tennessee. Além desses outros contratos subsequentes: Universidade de Utah, Pittsburgh e Columbia. Esses parceiros e utilizadores estão ilustrados na Figura 4.2.

Figura 4.2. Portadores de contratos e utilizadores da ITK: The Office of High Performace Computing and communications - NLM, General Eletric Corporate R&D, Kitware, Inc., Universidade da Carolina do Norte, Pennsylvania, Tennessee, Utah, Pittsburgh e Columbia.

Entre as aplicações clínicas, vale destacar os softwares: ITK SNAP [Guia ITK], BRASIL, *NeuroCad, Betir* [Oliveira *et al.*], dentre outros. As aplicações científicas são imensuráveis, com uma infinidade de trabalhos e artigos científicos que em suas técnicas para informática médica ou processamento digital de imagem incorporam o ITK em sua implementação.

4.2.4 Instalação

ITK é um conjunto de ferramentas e, como tal, uma vez instalado em seu computador não fornecerá um ícone para execução, pois consiste em um grande conjunto de bibliotecas que podem ser usadas para criar seus próprios aplicativos. Além do kit de ferramentas básicas, também inclui um extenso conjunto de exemplos e testes que introduzem conceitos ITK e mostram como usá-los em seus próprios projetos.

Alguns dos exemplos disponibilizados dependem de bibliotecas de terceiros, elas geralmente são instaladas separadamente. Para a criação inicial de projetos ITK, você pode querer ignorar essas bibliotecas extras e apenas compilar o próprio kit de ferramentas.

Para ser instalado, faz-se necessário um compilador para a linguagem C++ e o software de auxílio à compilação e instalação de pacotes *CMaker*. Dentre os compiladores suportados estão: *GCC 2.95 – Visual C++ 6.0, 7.0, 7.1 e/ou 8.0, Intel 7.1, 8.0, e/ou 9, IRIX CC, Borland 5.5 e Mac –gcc.*

CMaker é um software para projeção, construção, testes e instalação de pacotes de softwares. Foi desenvolvido pela *Kitware* para atender a demanda de compilação de projetos de código aberto e multi-plataformas como ITK e VTK. Este ambiente é

utilizado para controlar processos de compilação de pacotes de *software* para sistemas e compiladores que atendam a necessidade de cada usuário e desenvolvedor, gerando *makefiles* e espaços de trabalho versáteis e genéricos.

O processo de instalação do ITK consiste em instalar os pré-requisitos supracitados, realizar o download do código da biblioteca desejada e compilar través do ambiente gráfico do *CMaker*. Dessa maneira as aplicações poderão importar as bibliotecas instaladas e suas respectivas funcionalidades. A Figura 4.3 ilustra o ambiente gráfico *CMaker* na configuração e construção do ITK.

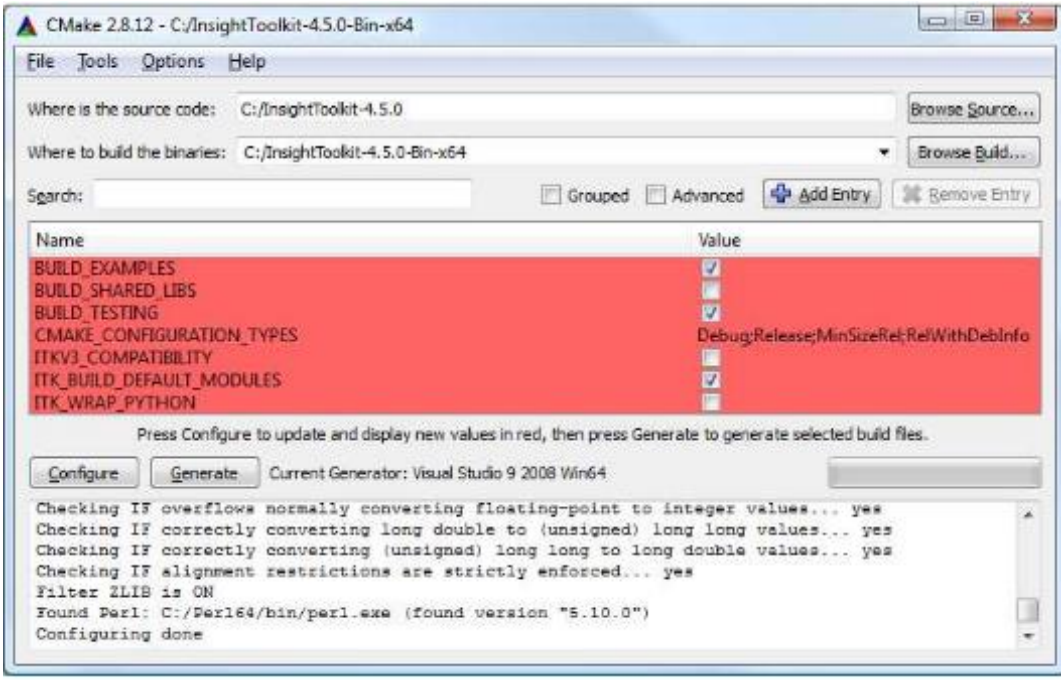


Figura 4.3. Interface gráfica CMaker para configuração e construção do ITK 4.5.0

4.3. Operações básicas

4.3.1. Leitura

No ITK existe dois tipos principais de classes para representar os dados, são elas*: itk::Image e itk::Mesh*. A classe *itk:Image* é utilizada para a representação de imagens com qualquer tipo de *pixel* e com diferentes dimensões. Essa também segue os princípios de programação genérica. Nesta seção será mostrado como são implementadas as operações básicas para leitura de imagem no ITK.

O ITK não impõe qualquer formato de arquivos específico, ele fornece uma estrutura que suporta uma variedade de formatos. E essas estruturas podem ser estendidas pelo usuário para suportar formatos novos.

A classe responsável pela leitura de imagem, fica localizada no início do fluxo de processamento, é conhecida como fonte de dados ou leitores, essa classe é a *itk::ImageFileReader*. Desta forma, o primeiro passo para realizar a leitura de uma imagem é incluir o seguinte cabeçalho no código:

#include "itkImageFileReader.h".

O próximo passo é escolher o tipo de pixel usado para representar a imagem que será lida e processada pelo fluxo de processamento. Um exemplo de código para imagem medica é mostrado abaixo:

```
typedef short PixelType;

const unsigned int Dimension = 2;

typedef itk::Image< PixelType, Dimension > ImageType;
```

A dimensão definida para a imagem no código acima, tem que ser mesma da imagem que será lida. Em seguida, deve ser criada uma instancia para a classe *itkImageFileReader*, essa instancia é parametrizada sobre o tipo da imagem. Como observa-se no código abaixo:

```
typedef itk::ImageFileReader< ImageType > ReaderType;
```

O próximo passo é criar um objeto para tipo *ReaderType* usando o método *New ()*, e atribuí-lo a um *itk::SmartPointer.*

```
ReaderType::Pointer reader = ReaderType::New();
```

Por fim, o objeto criado recebe a imagem, através do método *SetFileName()*, como mostrado no código abaixo:

```
 reader->SetFileName(inputFilename);
```

4.3.2 Escrita

A escrita é a etapa onde a imagem de saída do fluxo de processamento é salva. No ITK a escrita é gerenciada pela classe *itk::ImageFileWriter*. Assim, o primeiro passo para implementar a escrita será incluir o seguinte cabeçalho:

```
#include "itkImageFileWriter.h"
```

Assim como na leitura, para realizar a escrita é necessário escolher o tipo de pixel e a dimensão da imagem que será escrita, onde a dimensão da imagem que será escrita deve ser a mesma da imagem processada.

```
typedef short PixelType;

const unsigned int Dimension = 2;

typedef itk::Image< PixelType, Dimension > ImageType;
```

Em seguida deve ser criada uma instancia para a classe *itkImageFileWriter*. Como observa-se no código abaixo:

```
typedef itk::ImageFileWriter< ImageType > WriterType;
```

O próximo passo é criar um objeto do tipo *WriteType* usando o método *New ()* e atribui-lo a um *itk::SmartPointer*.

```
WriterType::Pointer writer = WriterType::New();
```

E por fim a imagem é escrita utilizando o método *SetFileName()*, como mostrado no código abaixo:

```
Writer->SetFileName( outputFilename );
```

4.3.3. Manipulação da matriz de pixels/voxels

As imagens no ITK, são transformadas em uma matriz, cuja a quantidade de linhas e colunas é definida pela quantidade de dimensões da imagem. Desta forma, o valor da armazenado na matriz representa um *pixel* da imagem.

Desta forma, cada valor da imagem é identificado e acessado através de um índice exclusivo. Esse índice é uma matriz de inteiros que define a posição do *pixel* ao longo de cada dimensão da imagem. *IndexType* é automaticamente definido pela imagem e pode ser acessado usando *o itk::Index.*

O código a seguir ilustra a declaração de uma variável de índice e a atribuição dos valores do *pixel* que será acessado, as seguintes linhas declaram uma instância do tipo de índice e inicializa o seu conteúdo para associá-lo com uma posição do *pixel* na imagem.

```
const ImageType::IndexType pixelIndex = {{27,29,37}}; // Position of {X,Y,Z}.
```

Após definir a posição do pixel que será acessado por meio do índice, é possível acessar o conteúdo do *pixel* na imagem. O método *getPixel ()* permite obter o valor do pixel. Como mostra o código abaixo:

```
ImageType::PixelType pixelValue = image->GetPixel( pixelIndex );
```

Existe também o método *setPixel ()* que permite definir o valor do pixel.

```
image->SetPixel( pixelIndex, pixelValue+1 );
```

Esses dois métodos são relativamente lentos e não devem ser utilizados em situações onde o acesso de alto desempenho é necessário. Os iteratores de imagem são os mecanismos adequados para acesso eficiente.

Iteratores são uma abstração de um ponteiro de memória, utilizados para acessar dados de forma iterativa. Através deles é possível percorrer todos os pixels de uma imagem de forma rápida, em imagens de N-dimensões. Os iteratores mais comuns são *ImageRegionIterato* e *ImageRegionConstIterator*. Que são usados para acessar os pixels da imagem.

4.4. Pré-Processamento

4.4.1. Filtros

Como podemos observar no item 4.2.2, que trata da arquitetura do ITK, o fluxo comum de operações envolvem a utilização dos filtros ou objetos de processamento, classes que recebem uma imagem, realizam determinado processamento e retornam outra imagem com a manipulação desejada. A Figura 4.4 ilustra o processamento de imagens com filtros ITK.

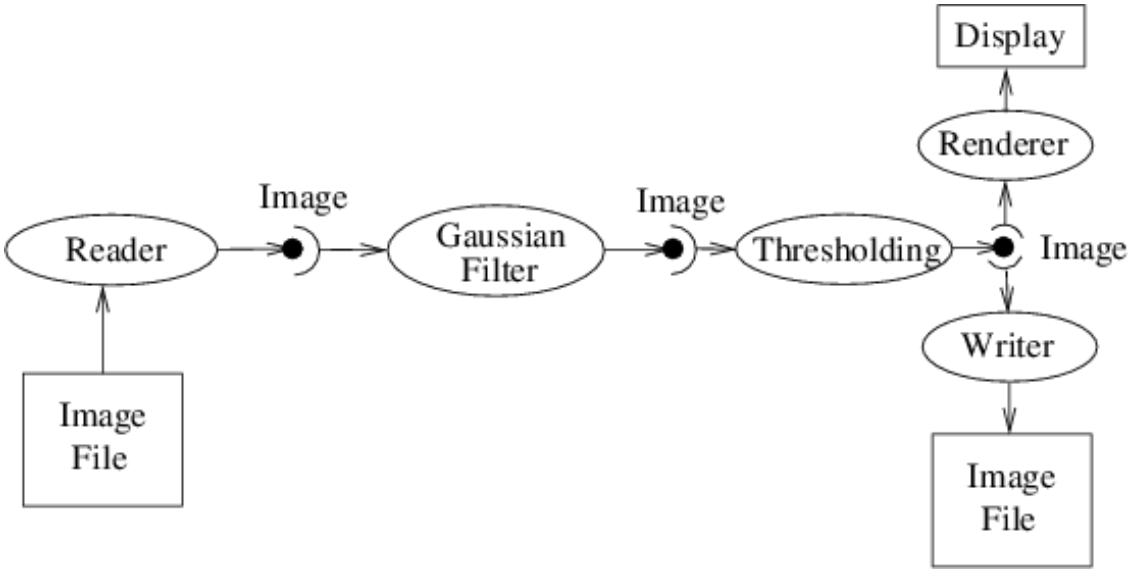


Figura 4.4. Ilustração de um processamento de imagem com filtros de ITK.

A partir desta lógica de processamento, os filtros são ferramentas essenciais à manipulação de imagens médicas, de modo que a partir da problemática trabalhada em cada objeto de estudo, deve-se, incialmente, selecionar quais filtros poderão ser utilizados, adequados ou implementados para sanar objetivos específicos.

Dada a relevância e complexidade dos filtros, [ZIEMER *et al*.], em sua apresentação do *ImageLab*, dividem os filtros em categorias, sendo elas: Filtragem por tons de cinza *(thresholding*), que manipulam valores de voxels da imagem; Filtros de suavização (*smoothing*), que buscam o melhoramento da imagem em determinado aspecto; Filtros de detecção de Borda (*edge Detection*), para ressaltar contornos; Filtros de Gradiente, que auxiliam na separação de regiões da imagem; Filtros de vizinha, que realizam a convolução de pixels baseada nos valores dos vizinhos; E Filtros de Segmentação,

4.4.2. Detectores de borda

A borda de regiões em uma imagem constitui um parâmetro significativo para sua compreensão ou manipulação. Através da detecção automática de bordas ou limites, pode-se obter a superfície de determinado objeto, conhecer seu perímetro e mensurar sua densidade. No contexto de imagens médicas, a superfície de uma lesão pode definir a quão impactante ou danosa ela pode ser, tornando essa informação imprescindível em diversas situações.

Para a extração automática de bordas, podemos encontrar filtros e módulos ITK específicos, que objetivam detectar limites entre regiões da imagem ou superfícies de possíveis estruturas. Dentre eles destacam-se:

- *BinaryContourImageFilter;*
- *LabelContourImageFilter;*
- *BinaryErodeImageFilter;*
- *SimpleContourExtractorImageFilter;*

- *CannyEdgeDetectionImageFilter.*

Os operadores e filtros citados perpassam por operações básicas em imagens binarias à analises complexas de superfícies.

Os operadores e filtros citados perpassam por operações básicas em imagens binarias à analises complexas de superfícies. Abaixo serão mostrados exemplos utilizando esses filtros. As figuras 4.5 e 4.6 mostram o código fonte e um exemplo de entrada e saída utilizando o filtro *BinaryContourImageFilter.*

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkImageFileReader.h"
#include "itkBinaryContourImageFilter.h"

typedef itk::Image<unsigned char, 2>  ImageType;
typedef itk::ImageFileWriter<ImageType> WriterType;

int main()
{
    ImageType::Pointer image;
    typedef itk::ImageFileReader<ImageType> ReaderType;
    ReaderType::Pointer reader =  ReaderType::New();
    reader->SetFileName("../entrada.png");
    reader->Update();
    image = reader->GetOutput();

    typedef itk::BinaryContourImageFilter <ImageType, ImageType > binaryContourImageFilterType;
    binaryContourImageFilterType::Pointer binaryContourFilter = binaryContourImageFilterType::New ();
    binaryContourFilter->SetInput(image);

    WriterType::Pointer writer = WriterType::New();
    writer->SetFileName("../saida.png");
    writer->SetInput(binaryContourFilter->GetOutput());
    writer->Update();

    return EXIT_SUCCESS;
}
```

Figura 4.5. Código fonte para o BinaryContourImageFilter;

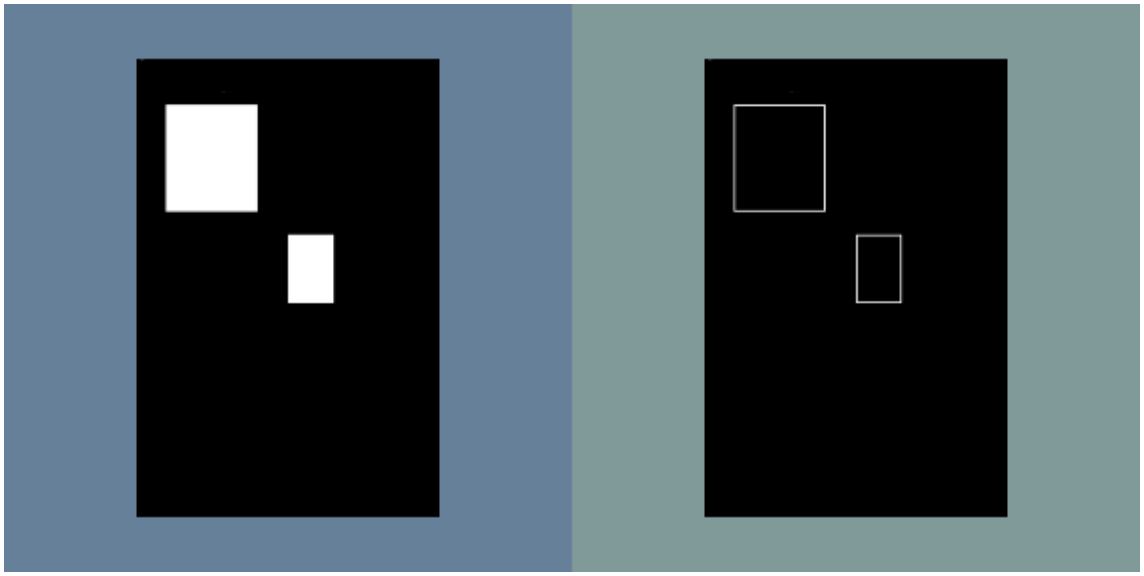


Figura 4.6. Imagem de entrada e saída para BinaryContourImageFilte

As Figuras 4.7 e 4.8 mostram o esquema para detecção de borda proposto por [Canny 1986].

```
#include "itkCastImageFilter.h"
#include "itkCannyEdgeDetectionImageFilter.h"

#include "QuickView.h"

int main(int argc, char *argv[])
{
  if(argc < 2)
    {
    std::cerr << "Usage: " << argv[0]
              << " Filename [variance lower_threshold upper_threshold]"
              << std::endl;
    return EXIT_FAILURE;
    }

  double variance = 2.0;
  double upperThreshold = 0.0;
  double lowerThreshold = 0.0;
  if (argc > 2)
    {
    variance = atof(argv[2]);
    }
  if (argc > 3)
    {
    lowerThreshold = atof(argv[3]);
    }
  if (argc > 4)
    {
    upperThreshold = atof(argv[4]);
    }

  typedef itk::Image<double, 2>  DoubleImageType;
```

```
  typedef itk::ImageFileReader<DoubleImageType> ReaderType;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  typedef itk::CannyEdgeDetectionImageFilter <DoubleImageType, DoubleImageType>
    CannyEdgeDetectionImageFilterType;

  CannyEdgeDetectionImageFilterType::Pointer cannyFilter
    = CannyEdgeDetectionImageFilterType::New();
  cannyFilter->SetInput(reader->GetOutput());
  cannyFilter->SetVariance( variance );
  cannyFilter->SetUpperThreshold( upperThreshold );
  cannyFilter->SetLowerThreshold( lowerThreshold );

  QuickView viewer;
  viewer.AddImage<DoubleImageType>(reader->GetOutput());
  viewer.AddImage<DoubleImageType>(cannyFilter->GetOutput());
  viewer.Visualize();

  return EXIT_SUCCESS;
}
```

Figura 4.7. Código fonte, para esquema de detecção de bordas proposto por [Canny 1986].

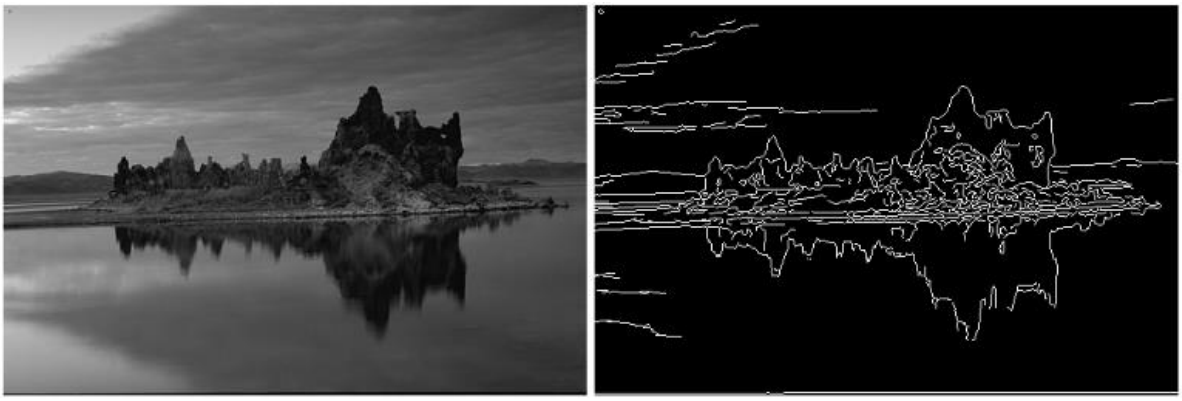


Figura 4.8. Entrada e saída com o esquema de detecção de bordas proposto por Canny (1986). Fonte: Adaptado de [LOURENÇO, 2011]

4.4.3. Operações morfológicas

Muitas doenças possuem características peculiares e significativas em suas feições e estruturas. Uma característica comum do câncer, por exemplo, é o crescimento desordenado das células doentes, fazendo com que seja constituída uma superfície com forma atípica e irregular, mas que descreve o desenvolvimento da doença e, consequentemente, é fundamental para seu entendimento.

Para a análise das peculiaridades de forma de estruturas presentes em imagens médicas, utiliza-se a morfologia matemática, um conceito fundamentado na teoria dos conjuntos que consiste na análise das características geométricas e topográficas de uma região.

A morfologia matemática é uma teoria proposta por Georges Matheron e Jean Serra. No processamento digital de imagens pode ser usada para diversas finalidades, como realce, filtragem, segmentação, detecção de superfícies, afinamento, dentre outras. O objetivo da técnica, baseada na teoria dos conjuntos, é mensurar características morfológicas, ou extrair informações espaciais a partir de operadores chamados de elementos estruturantes. Dentre as operações, se destacam: translação, reflexão, erosão, dilatação, abertura, fechamento, afinamento, transformadas, fecho convexo, dentre outros. A tabela 1 demonstra alguns destes operadores e seu contexto [Filho e Neto 1999].

Tabela 1. Principais algoritmos morfológicos

Operação	Equação	Descrição
Translação	$(A)_x = \{c \mid c = a + x, para \quad a \in A\}$	Translata a origem de A para o ponto x.
Reflexão	$\hat{B} = \{x \mid x = -b, \quad para \quad b \in B\}$	Reflete todos os elementos de B em relação à origem deste conjunto.

Complemento	$A^c = \{x \mid x \notin A\}$	Conjunto de pontos que não estão em A.
Diferença	$A - B = \{x \mid x \in A, x \notin B\} = A \cap B^c$	Conjunto de pontos que pertencem a A, porém não a B.
Dilatação	$A \oplus B = \{x \mid (\hat{B})_x \cap A \neq \varnothing\}$	'Expande' a fronteira de A. (I)
Erosão	$A \ominus B = \{x \mid (B)_x \subseteq A\}$	'Contrai' a fronteira de A. (I)
Abertura	$A \circ B = (A \ominus B) \oplus B$	Suaviza contornos, quebra istmos estreitos e elimina pequenas ilhas e picos agudos. (I)
Fechamento	$A \bullet B = (A \oplus B) \ominus B$	Suaviza contornos, une quebras estreitas e golfos longos e delgados e elimina pequenos orifícios. (I)

Na biblioteca ITK há um gama de bibliotecas complementares e módulos que se concentram na morfologia matemática, possibilitando a implementação dos operadores, a extração de características de forma e o desenvolvimento de novas técnicas ou filtros para esta finalidade. Se destacam:

- *Neighborhood Iterators;*
- *MorphologyImageFilter;*
- *binaryAttributeMorphology;*
- *BinaryContourImageFilter;*
- *LabelGeometryImageFilter;*
- *itkReconstructionByDilationImageFilter;*
- *itkHConvexImageFilter;*
- *BinaryErodeImageFilter*.
- *BinaryDilateImageFilter;*
- *BinaryMorphologicalOpeningImageFilter;*
- *SubtractImageFilter;*
- *BinaryMorphologicalClosingImageFilter.*

Entre as Figuras 4.9 e 4.16 são demonstradas as implementações e respectivas entradas e saídas das principais operações morfológicas (dilatação, erosão, abertura e fechamento), através de Filtros ITK.

```
#include "itkBinaryErodeImageFilter.h"
#include "itkBinaryBallStructuringElement.h"

...

typedef itk::BinaryBallStructuringElement<ImageType::PixelType, 2> StructuringElementType;
StructuringElementType structuringElement;
structuringElement.SetRadius(radius);
structuringElement.CreateStructuringElement();

typedef itk::BinaryErodeImageFilter <ImageType, ImageType, StructuringElementType> BinaryErodeImageFilterType;

BinaryErodeImageFilterType::Pointer erodeFilter = BinaryErodeImageFilterType::New();
erodeFilter->SetInput(reader->GetOutput());
erodeFilter->SetKernel(structuringElement);
saida = erodeFilter->GetOutput();
```

Figura 4.9. Código fonte para a operação de Erosão.

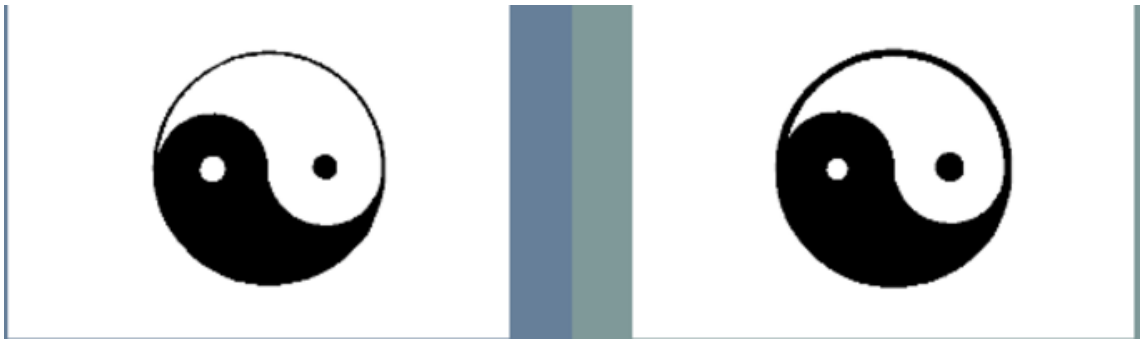


Figura 4.10. Exemplo de imagem de entrada e de saída com o operador de Erosão.

```
#include "itkBinaryDilateImageFilter.h"
#include "itkBinaryBallStructuringElement.h"

...

unsigned int radius = 2;

typedef itk::BinaryBallStructuringElement<ImageType::PixelType,2>StructuringElementType;
StructuringElementType structuringElement;
structuringElement.SetRadius(radius);
structuringElement.CreateStructuringElement();

typedef itk::BinaryDilateImageFilter <ImageType, ImageType, StructuringElementType>
      BinaryDilateImageFilterType;

BinaryDilateImageFilterType::Pointer dilateFilter
      = BinaryDilateImageFilterType::New();
dilateFilter->SetInput(reader->GetOutput());
dilateFilter->SetKernel(structuringElement);

saida = dilateFilter->GetOutput();
```

Figura 4.11. Código fonte para a operação de dilatação.

Figura 4.12. Exemplo de imagem de entrada e de saída com o operador de dilatação.

```
#include "itkBinaryMorphologicalOpeningImageFilter.h"
#include "itkBinaryBallStructuringElement.h"
#include "itkSubtractImageFilter.h"

...

typedef itk::BinaryBallStructuringElement<ImageType::PixelType, ImageType::ImageDimension>
          StructuringElementType;
StructuringElementType structuringElement;
structuringElement.SetRadius(radius);
structuringElement.CreateStructuringElement();

typedef itk::BinaryMorphologicalOpeningImageFilter <ImageType, ImageType, StructuringElementType>
      BinaryMorphologicalOpeningImageFilterType;
BinaryMorphologicalOpeningImageFilterType::Pointer openingFilter
      = BinaryMorphologicalOpeningImageFilterType::New();
openingFilter->SetInput(image);
openingFilter->SetKernel(structuringElement);
openingFilter->Update();

typedef itk::SubtractImageFilter<ImageType> SubtractType;
SubtractType::Pointer diff = SubtractType::New();
diff->SetInput1(image);
diff->SetInput2(openingFilter->GetOutput());

saida1 = openingFilter->GetOutput();
saida2 = diff->GetOutput();
```

Figura 4.13. Código fonte para a operação de Abertura.

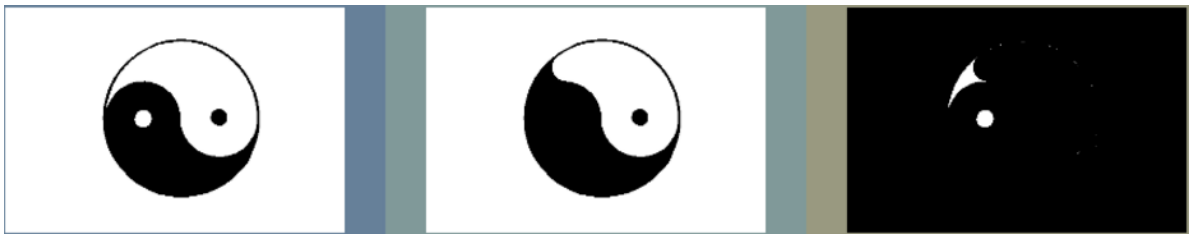


Figura 4.14. Exemplo de imagem de entrada e de saída com o operador de Abertura. 1) original. (2) Abertura binária com radius = 20. (3) Abertura binária original.

```
#include "itkBinaryMorphologicalClosingImageFilter.h"
#include "itkBinaryBallStructuringElement.h"
#include "itkSubtractImageFilter.h"

...

typedef itk::BinaryBallStructuringElement<ImageType::PixelType, ImageType::ImageDimension>
          StructuringElementType;
StructuringElementType structuringElement;
structuringElement.SetRadius(radius);
structuringElement.CreateStructuringElement();

typedef itk::BinaryMorphologicalClosingImageFilter <ImageType, ImageType, StructuringElementType>
      BinaryMorphologicalClosingImageFilterType;
BinaryMorphologicalClosingImageFilterType::Pointer closingFilter
      = BinaryMorphologicalClosingImageFilterType::New();
closingFilter->SetInput(image);
closingFilter->SetKernel(structuringElement);
closingFilter->Update();

typedef itk::SubtractImageFilter<ImageType> SubtractType;
SubtractType::Pointer diff = SubtractType::New();
diff->SetInput1(closingFilter->GetOutput());
diff->SetInput2(image);
saida1 = closingFilter->GetOutput();
saida2 = diff->GetOutput();
```

Figura 4.15. Código fonte para a operação de Fechamento.

Figura 4.16. Exemplo de imagem de entrada e de saída com o operador de Fechamento. 1) original. (2) fechamento binário com radius = 20. (3) fechamento binário original.

4.4.4. Operações lógicas

Uma imagem digital é, também, representada por uma matriz de valores inteiros, ou *pixels*, cujo valor reflete binários, tons de cinza ou cor, a depender do esquema utilizado. A partir disso, como já apresentado, pode-se manipular uma imagem, através da matriz de valores de seus *pixels*, com múltiplos operadores, dentre eles lógicos, aritméticos, morfológicos, dentre outros.

Operações binárias, ou booleanas, podem ser aplicadas neste contexto, com imagens binárias ou em tons de cinza, e acontecem entre *pixels* em coordenadas homólogas de 2 matrizes (imagens), que são relacionados (*And, Or, Xor*, por exemplo) para a obtenção de uma terceira matriz, consequentemente uma imagem resultante das múltiplas operações [Filho e Neto 1999].

As Figuras 4.17, 4.18, 4.19 e 4.20, exemplificam e ilustram estas operações. Em ITK existem classes que possibilitam essas operações, são elas:

- *itkOrImageFilter;*
- *itkAndImageFilter;*
- *XorImageFilter*;

- *BinaryNotImageFilter*.

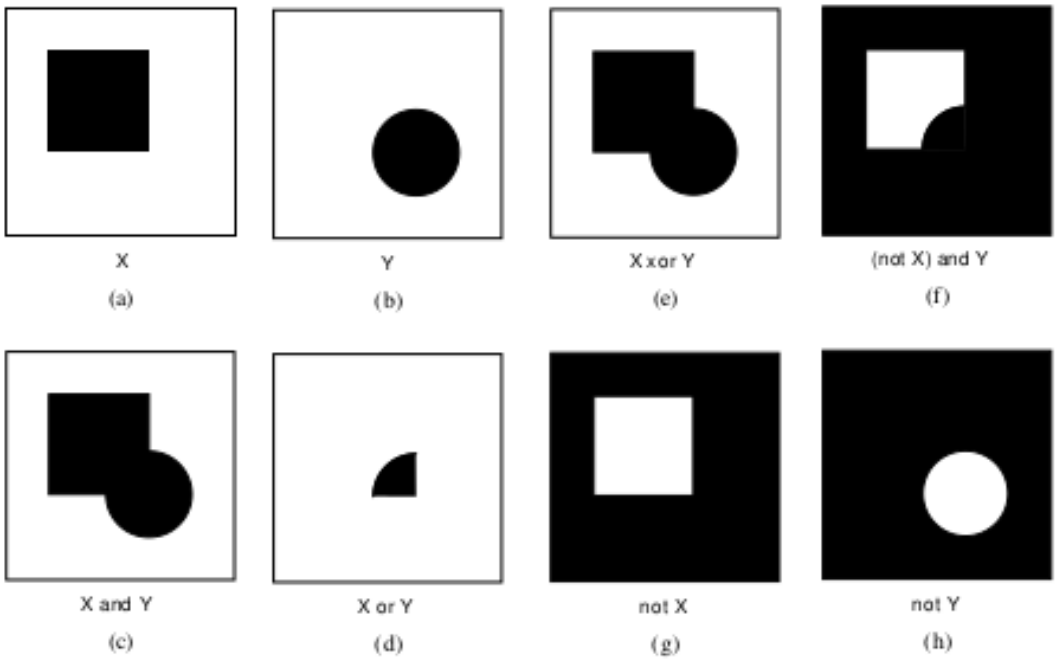


Figura 4.17. Operadores lógicos em imagens binárias. Fonte: Adaptado de [Filho e Neto 1999].

Figura 4.18. **Operador "And" para duas figuras em** tons de cinza (a e b), sendo c = a^b.

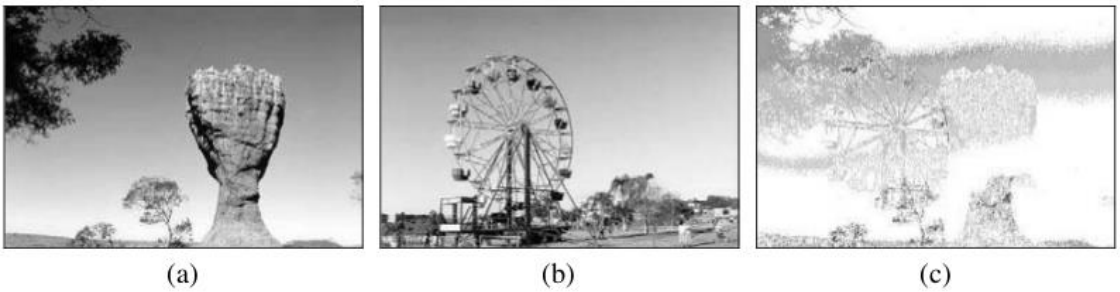


Figura 4.19. Operador **"Or" para duas figuras em tons de cinza (a e b), sendo c** = a (OR) b. Fonte: Adaptado de [Filho e Neto 1999].

(a) (b) (c)

Figura 4.20. **Operador "Xor" para duas figuras em tons de cinza (a e b), sendo c** = a (XOR) b. Fonte: Adaptado de [Filho e Neto 1999]

(a) (b)

Figura 4.21. **Operador unário "Not" para uma figura em tons de cinza, sendo "a" a entrada e "b" a saída do operador. F**onte: Adaptado de [Filho e Neto 1999].

As figuras 4.22, 4.23, 4.24 e 4.25 mostram os códigos fontes das operações *and, or, xor e not.*

```
#include "itkAndImageFilter.h"
 ...

typedef itk::AndImageFilter <ImageType> AndImageFilterType;
AndImageFilterType::Pointer andFilter = AndImageFilterType::New();
andFilter->SetInput(0, image1);
andFilter->SetInput(1, image2);
andFilter->Update();
saida = andFilter->GetOutput();
```

Figura 4.22. Código fonte para a operação and.

```
#include "itkOrImageFilter.h"
...

typedef itk::OrImageFilter <ImageType> OrImageFilterType;
OrImageFilterType::Pointer orFilter = OrImageFilterType::New();
orFilter->SetInput(0, image1);
orFilter->SetInput(1, image2);
orFilter->Update();
saida = orFilter->GetOutput();
```

Figura 4.23. Código fonte para a operação or.

```
#include "itkXOrImageFilter.h"
...

typedef itk::XorImageFilter <ImageType> XorImageFilterType;
XorImageFilterType::Pointer xorFilter = XorImageFilterType::New();
xorFilter->SetInput1(image1);
xorFilter->SetInput2(image2);
xorFilter->Update();
saida = xorFilter->GetOutput();
```

Figura 4.24. Código fonte para a operação xor.

```
#include "itkBinaryNotImageFilter.h"
...

typedef itk::BinaryNotImageFilter <ImageType> BinaryNotImageFilterType;
BinaryNotImageFilterType::Pointer binaryNotFilter = BinaryNotImageFilterType::New();
binaryNotFilter->SetInput(image);
binaryNotFilter->Update();
saida = andFilter->binaryNotFilter->Output();
```

Figura 4.25. Código fonte para a operação not.

4.4.5. Esqueletização

Os esqueletos de uma forma geométrica são constituídos pelas coordenadas que representam os centros dos círculos que tocam pelo menos 2 pontos de sua borda [SAMPAIO 2015], como demonstrado na Figura 4.26.

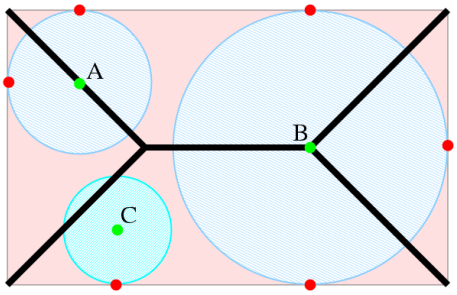


Figura 4.26 **Ilustração do esqueleto de um retângulo, onde os pontos "A" e "B" peretencem ao esqueleto e "C" nã**o. Fonte: Adaptado de [SAMPAIO 2015].

Portanto, o esqueleto de um objeto representa as características de sua forma e superfície num conjunto reduzido de pontos de coordenadas, possibilitando compreender rapidamente o volume do objeto e o comportamento de sua superfície para cada região de seu esqueleto. Pode-se então extrair como característica o tamanho do esqueleto de determinado objeto, como também sua relação com outros parâmetros do objeto, como área, *Feret* máximo, *Feret* mínimo, dentre outras.

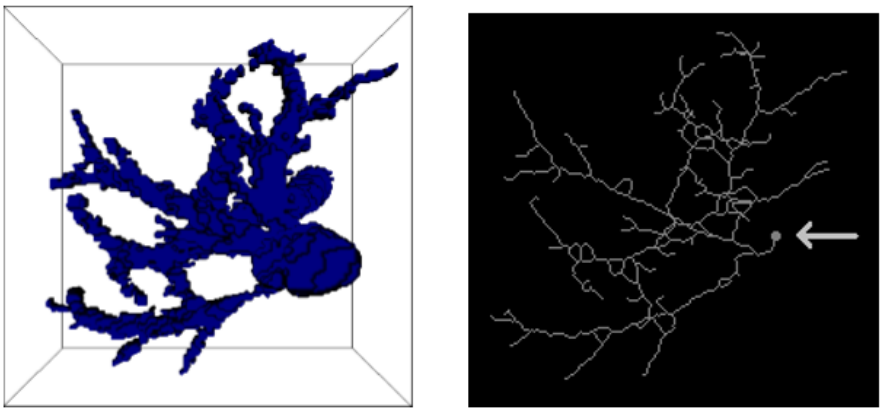
[Homann 2007] propõe uma biblioteca para esqueletização de objetos tridimencionais a partir do algoritmo proposto por [Lee et al.], objetivando substituir a biblioteca nativa *itkBinaryThinningImageFilter2D*. A biblioteca proposta é denominada por *itkBinaryThinningImageFilter3D*, que necessita de instalação específica para importação. Essa nova implementação possibilita a manipulação de diversas estruturas

trabalhas na informática médica, que são geralmente representadas por imagens volumétricas. Um exemplo de entrada e saída deste filtro é ilustrado nas figuras 4.27 e 4.28 apresentada por seu autor.

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"

#include "itkConnectedThresholdImageFilter.h"
#include "itkImageRegionIterator.h"
#include "itkBinaryThinningImageFilter3D.h"

#include <iostream>
#include <stdlib.h>   // for atoi()
using namespace std;

int main(int argc, char* argv[])
{
  // Verify the number of parameters in the command line
  if( argc <= 2 )
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " inputImageFile outputImageFile" << std::endl;
    return EXIT_FAILURE;
  }
  char* infilename  = argv[1];
  char* outfilename = argv[2];

  const   unsigned int Dimension = 3;
  typedef signed short PixelType;   // must be signed for CT since Hounsfield units can be < 0
  typedef itk::Image< PixelType, Dimension > ImageType;

  // Read image
  typedef itk::ImageFileReader< ImageType > ReaderType;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName( infilename );
```

```
  try
  {
    reader->Update();
  }
  catch (itk::ExceptionObject &ex)
  {
    std::cout << ex << std::endl;
    return EXIT_FAILURE;
  }
  cout << infilename << " sucessfully read." << endl;
  // Define the thinning filter
  typedef itk::BinaryThinningImageFilter3D< ImageType, ImageType > ThinningFilterType;
  ThinningFilterType::Pointer thinningFilter = ThinningFilterType::New();
  thinningFilter->SetInput( reader->GetOutput() );
  thinningFilter->Update();
  // output to file
  typedef itk::ImageFileWriter< ImageType > WriterType;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput( thinningFilter->GetOutput() );
  writer->SetFileName( outfilename );
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject &ex)
  {
    std::cout << ex << std::endl;
    return EXIT_FAILURE;
  }
  cout << outfilename << " sucessfully written." << endl;
  cout << "Program terminated normally." << endl;
  return EXIT_SUCCESS;
}
```

Figura 4.27. Código fonte para extração do esqueleto



a) b)

Figura 4.28. a) Rederização de superfície volumétrica, ou 3D, submetida ao filtro. b) Imagem de saída, ou respectivo esqueleto projetado. Fonte: Adaptado de [Homann, 2007].

4.5. Algoritmos de Segmentação

4.5.1. Clusterizadores

Cluster consiste em agrupar dados em classes ou grupos, de forma que objetos dentro de um mesmo grupo tenham alta similaridade, e objetos pertencentes a grupos diferentes sejam distintos. Para medir a similaridade entre os objetos analisados são utilizadas diversas técnicas de medida, normalmente baseadas em medidas de distância.

Segmentação de imagens usando clusterizadores se dá pelo agrupamento de *pixels* em regiões de acordo com a similaridade entre eles, desta forma obtêm-se a região de interesse na imagem que serão utilizadas nas fases posteriores do processamento. Entre os algoritmos de agrupamento mais comuns está o *K-means.*

O k-*mens* que é uma técnica de aprendizado não supervisionado, consiste em dividir um conjunto de objetos em grupos, de forma que os objetos que pertencem ao mesmo grupo, apresentam alto de grau de similaridade, e objetos que pertencem grupos diferente não são similares. Desta forma, o *k-means* é usado para segmentação de imagens, onde tem-se como saída uma imagem segmentada em regiões, assim podem ser obtidas apenas as regiões de interesse que serão usadas nas fases seguintes do processamento.

O método tem como entrada um conjunto de objetos, e é indicada quantidade de grupos que o método agrupara os objetos. As etapas abaixo descrevem o funcionamento do *k-means.*

1- Incialmente, tem se com entrada o conjunto de objetos e quantidade de grupos que se deseja obter. A partir disso são escolhidos dados que serão os centroides, é escolhido um para cada grupo, os centroides serão as sementes.

2- Em seguida, é calculada a distância de cada objeto para cada centroide.

3- Nesta etapa, os objetos serão agrupados no grupo ao qual apresentem a menor distancia para o centroide.

4- Agora, os centroides serão redefinidos, é feita a etapa de 3 de novo.

5- A etapa 4 será repetida até que não haja mudanças nos grupos.

Um exemplo de segmentação usando algoritmo *k- means,* com 5 clusters, no ITK é mostrado a seguir. As figuras 4.29, 4.30 e 4.31, mostram o código fonte, uma imagem de entrada e uma imagem de saída do método.

```
#include "stdio.h"
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkGDCMImageIO.h"
#include "itkMinimumMaximumImageCalculator.h"
#include "itkTranslationTransform.h"
#include "itkResampleImageFilter.h"
#include "itkScalarImageKmeansImageFilter.h"
#include "itkOtsuMultipleThresholdsImageFilter.h"
using namespace  itk;
using namespace  std;
typedef itk::GDCMImageIO ImageIO;
typedef itk::Image<short,3> ImageType;
typedef itk::ImageFileReader<ImageType> ReaderType;
typedef itk::Image<short,3> ImageType3D;
typedef itk::ImageFileWriter<ImageType3D> WriterType;

int getMinimum(ImageType::Pointer image){
    typedef MinimumMaximumImageCalculator <ImageType> ImageCalculatorFilterType;
    ImageCalculatorFilterType::Pointer imageCalculatorFilter;
    imageCalculatorFilter = ImageCalculatorFilterType::New ();
    imageCalculatorFilter->SetImage(image);
    imageCalculatorFilter->Compute();
    return imageCalculatorFilter->GetMinimum();
}

int getMaximum(ImageType3D::Pointer image){
    typedef MinimumMaximumImageCalculator <ImageType3D> ImageCalculatorFilterType;
    ImageCalculatorFilterType::Pointer imageCalculatorFilter;
    imageCalculatorFilter = ImageCalculatorFilterType::New ();
    imageCalculatorFilter->SetImage(image);
    imageCalculatorFilter->Compute();
    return imageCalculatorFilter->GetMaximum();
}

ImageType3D::Pointer readerImage(string file){
    ImageIO::Pointer imgIO = ImageIO::New();
    ReaderType::Pointer reader = ReaderType::New();
    reader->SetFileName(file);
    reader->SetImageIO(imgIO);
    reader->Update();
    return reader->GetOutput();
}

ImageType3D::Pointer KmenasClutering(ImageType3D::Pointer image, int numCluster){

    typedef itk::ScalarImageKmeansImageFilter<ImageType3D,ImageType3D> KMeansFilterType;
    KMeansFilterType::Pointer kmeansImgFilter = KMeansFilterType::New();
    kmeansImgFilter->SetInput(image);
    kmeansImgFilter->SetUseNonContiguousLabels(true);

    int max, min;

    max = getMaximum(image);
    min = getMinimum(image);

    typedef itk::OtsuMultipleThresholdsImageFilter <ImageType3D, ImageType3D> FilterType;
```

```
    FilterType::Pointer otsuFilter = FilterType::New();
    otsuFilter->SetInput(image);
    otsuFilter->SetNumberOfThresholds(numCluster);
    otsuFilter->Update();
    FilterType::ThresholdVectorType thresholds = otsuFilter->GetThresholds();

    for (unsigned int i = 0; i < thresholds.size(); i++)
        kmeansImgFilter->AddClassWithInitialMean(itk::NumericTraits<FilterType::InputPixelType>::PrintType(thresholds[i]));

    kmeansImgFilter->Update();
    return kmeansImgFilter->GetOutput();
}
int main(){

    stringstream path;
    path << "C:/1.dcm";
    ImageType3D::Pointer image = readerImage(path.str());
    int numCluster=5;

    //ImageType3D::Pointer saida = KmenasClutering(image, 2);
    typedef itk::ScalarImageKmeansImageFilter<ImageType3D,ImageType3D> KMeansFilterType;
    KMeansFilterType::Pointer kmeansImgFilter = KMeansFilterType::New();
    kmeansImgFilter->SetInput(image);
    kmeansImgFilter->SetUseNonContiguousLabels(true);

    int max, min;
    max = getMaximum(image);
    min = getMinimum(image);

    typedef itk::OtsuMultipleThresholdsImageFilter <ImageType3D, ImageType3D> FilterType;
    FilterType::Pointer otsuFilter = FilterType::New();
    otsuFilter->SetInput(image);


    otsuFilter->SetNumberOfThresholds(numCluster);
    otsuFilter->Update();
    FilterType::ThresholdVectorType thresholds = otsuFilter->GetThresholds();

    for (unsigned int i = 0; i < thresholds.size(); i++)
        kmeansImgFilter->AddClassWithInitialMean(itk::NumericTraits<FilterType::InputPixelType>::PrintType(thresholds[i]));
    kmeansImgFilter->Update();

    typedef KMeansFilterType::OutputImageType  OutputImageType;
    typedef itk::ImageFileWriter< OutputImageType > WriterType;
    WriterType::Pointer writer = WriterType::New();
    writer->SetInput( kmeansImgFilter->GetOutput() );
    stringstream saida;
    saida << "C:/Users/Alexandre/s1.dcm";
    writer->SetFileName(saida.str());

    try
    {
        writer->Update();
    }
    catch( itk::ExceptionObject & excp )
    {
        std::cerr << "Problem encountered while writing ";
        std::cerr << " image file : " << saida.str() << std::endl;
        std::cerr << excp << std::endl;
        system("pause");
        return EXIT_FAILURE;

    }
```

Figura 4.29. Código fonte para k-mens

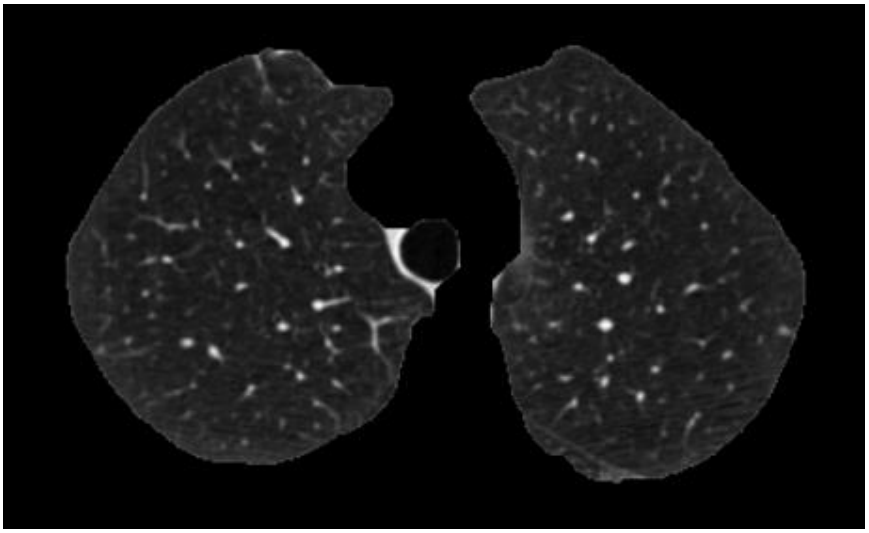


Figura 4.30. Imagem de entrada para k-mens

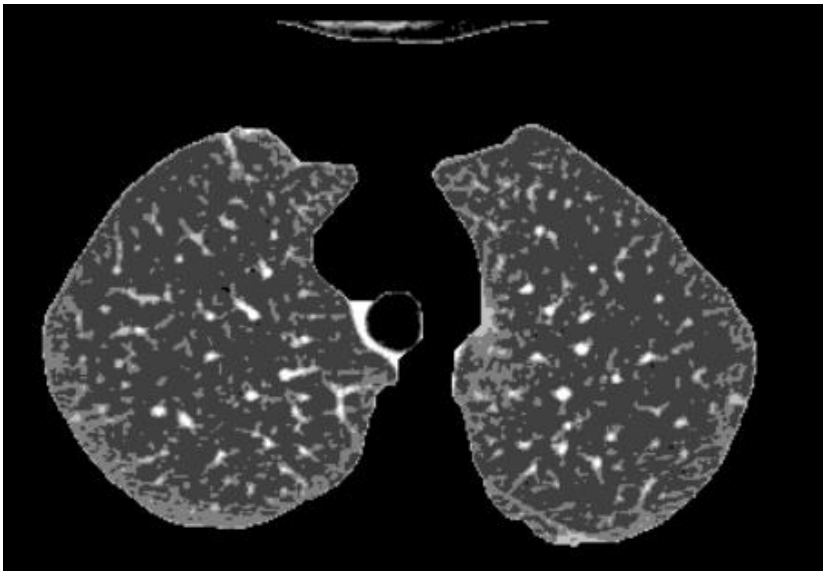


Figura 4.31. Imagem de saída para k-mens

4.5.2. Limiarização

Limiarização é uma abordagem utilizada para segmentação de imagens, essa técnica consiste em separar a imagem em duas classes fundo e objeto. Para isso é definido um limiar, que será utilizado para agrupar os *pixels* da imagem em uma das classes. Desta forma a escolha do limiar a adequado é essencial.

Matematicamente a limiarização pode ser definida como:

$$g(x,y) = \begin{cases} objeto\ se\ f(x,y) > T \\ fundo\ \ se\ f(x,y) < T \end{cases} \qquad (1)$$

Onde *f (x, y)* é a imagem de entrada, *T* é o valor do limiar e *g (x, y)* é a imagem de saída (limiarizada).

Os métodos de limiarização, utilizam geralmente dois tipos de limiar, um limiar global que aplicado para toda a imagem. Ou por mais de limiar que são aplicados em regiões diferentes da imagem. Um exemplo de método que utiliza um limiar global é o *Otsu*.

O método de *Otsu* é um algoritmo de limiarização que a partir de uma imagem em tons de cinza, determina o valor ideal de um limiar que separe os objetos e o fundo da imagem em dois *clusters*, atribuindo a cor branca ou preta para cada um deles. Devido à essa característica, funcionar especialmente bem para casos de imagens com histogramas bimodais, podendo ser divididas adequadamente com um único valor.

O método busca determinar um limiar de forma a maximizar a variância entre classes. A operação de limiarização consiste no particionamento dos *pixel*s de uma imagem com n- níveis de cinza em duas classes, $C0$ e $C1$, que representam o objeto e o fundo, ou vice-versa, sendo que esta partição se dará no nível de cinza t que representa o limiar ideal.

Para encontrar o limiar ideal, é necessário minimiza a variância intraclasse, definida como uma soma ponderada das variâncias das duas classes, através da formula mostra abaixo:

$$\sigma_b^2\,(t) = \sigma^2 - \sigma_w^2(t) = w_1(t)w_2(t)[\mu_2(t) - \mu_2(t)]^2 \qquad (2)$$

Para minimizar a variância intraclasse o método *otsu* maximiza a variância interclasse, através da seguinte formula.

$$\sigma_b^2\,(t) = \sigma^2 - \sigma_w^2(t) = w_1(t)w_2(t)[\mu_2(t) - \mu_2(t)]^2 \qquad (3)$$

As figuras 4.32 e 4.33 mostram o código fonte e um exemplo de entrada e saída de segmentação imagens no ITK usando o *otsu*.

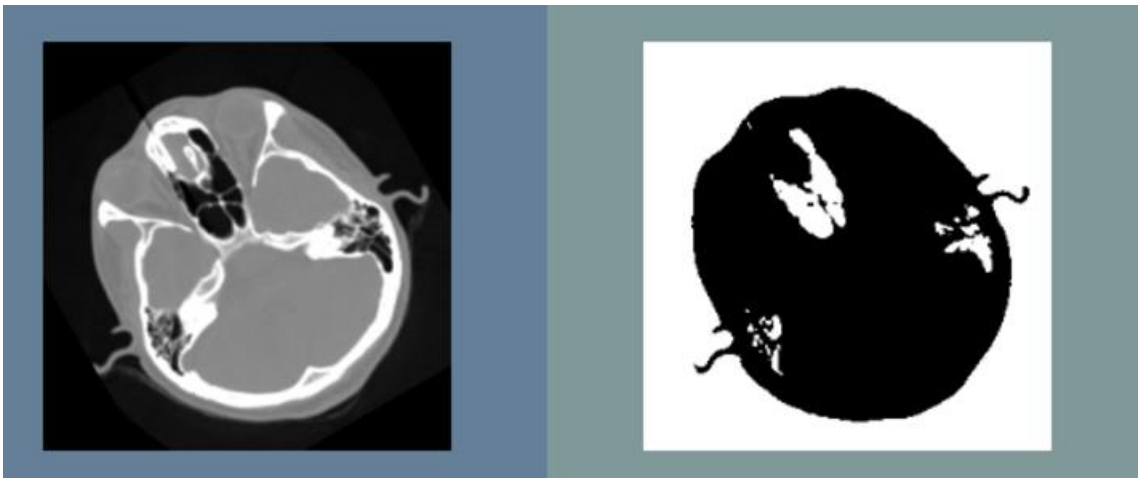


Figura 4.32. Exemplo de entrada e saída com método Otsu

```
#include "itkImage.h"
#include "itkOtsuThresholdImageFilter.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkImageRegionIterator.h"
#include "itkNumericTraits.h"

#include "itksys/SystemTools.hxx"
#include <sstream>

namespace
{
typedef unsigned char             PixelType;
typedef itk::Image<PixelType, 2>  ImageType;
typedef itk::ImageFileWriter<ImageType> WriterType;

}

int main()
{
    ImageType::Pointer image;
    typedef itk::ImageFileReader<ImageType> ReaderType;
    ReaderType::Pointer reader =  ReaderType::New();
    reader->SetFileName("C:/entrada.png");
    reader->Update();
    image = reader->GetOutput();

    typedef itk::OtsuThresholdImageFilter <ImageType, ImageType> FilterType;
    FilterType::Pointer otsuFilter= FilterType::New();
    otsuFilter->SetInput(image);
    otsuFilter->Update();
    WriterType::Pointer writer = WriterType::New();
    writer->SetFileName("C:/Users/Alexandre/saida.png");
    writer->SetInput(otsuFilter->GetOutput());
    writer->Update();

    return EXIT_SUCCESS;
}
```

Figura 4.33. Código fonte do método Otsu

4.5. 3. Crescimento de Região

A segmentação baseada em região é um método praticamente imune a ruídos, pela razão de se usar um critério de homogeneidade para a região de interesse. Esse critério de homogeneidade pode ser um limiar com base no nível de cinza, a cor ou a textura, forma, etc. [NETTO 2010]. O crescimento de região busca características semelhantes em *pixels* ou *voxels* próximos. O ponto de partida para o crescimento se dá a partir de determinados *pixels* ou *voxels* específicos, chamados de sementes. A técnica consiste nas seguintes etapas:

- Escolha das sementes (*voxe*l ou *pixel*);
- Escolha de um limiar que separará as regiões;
- E por fim, o crescimento das regiões;

A escolha das sementes é feita baseando-se na natureza do problema. A escolha destes pontos é importante, logo, as regiões crescerão ao redor dos mesmos. A escolha do limiar constitui um passo importante no processo, o valor a ser escolhido deve representar a diferença de intensidade desejável para cada uma das regiões.

Sendo assim o crescimento das regiões em si, constitui-se de fazer o agrupamento de *pixels* por similaridade em algum critério como os de intensidade de cinza, textura, entre outros mencionados anteriormente.

No âmbito tridimensional não é diferente, os parâmetros, as propriedades e a execução dos passos são os mesmos, porém como mostrado em [Netto 2010] é de extrema importância definir um intervalo de fatias para atuação da técnica, dessa forma agiria como um controle no processo de crescimento. A Figura 4.34 ilustra a técnica.

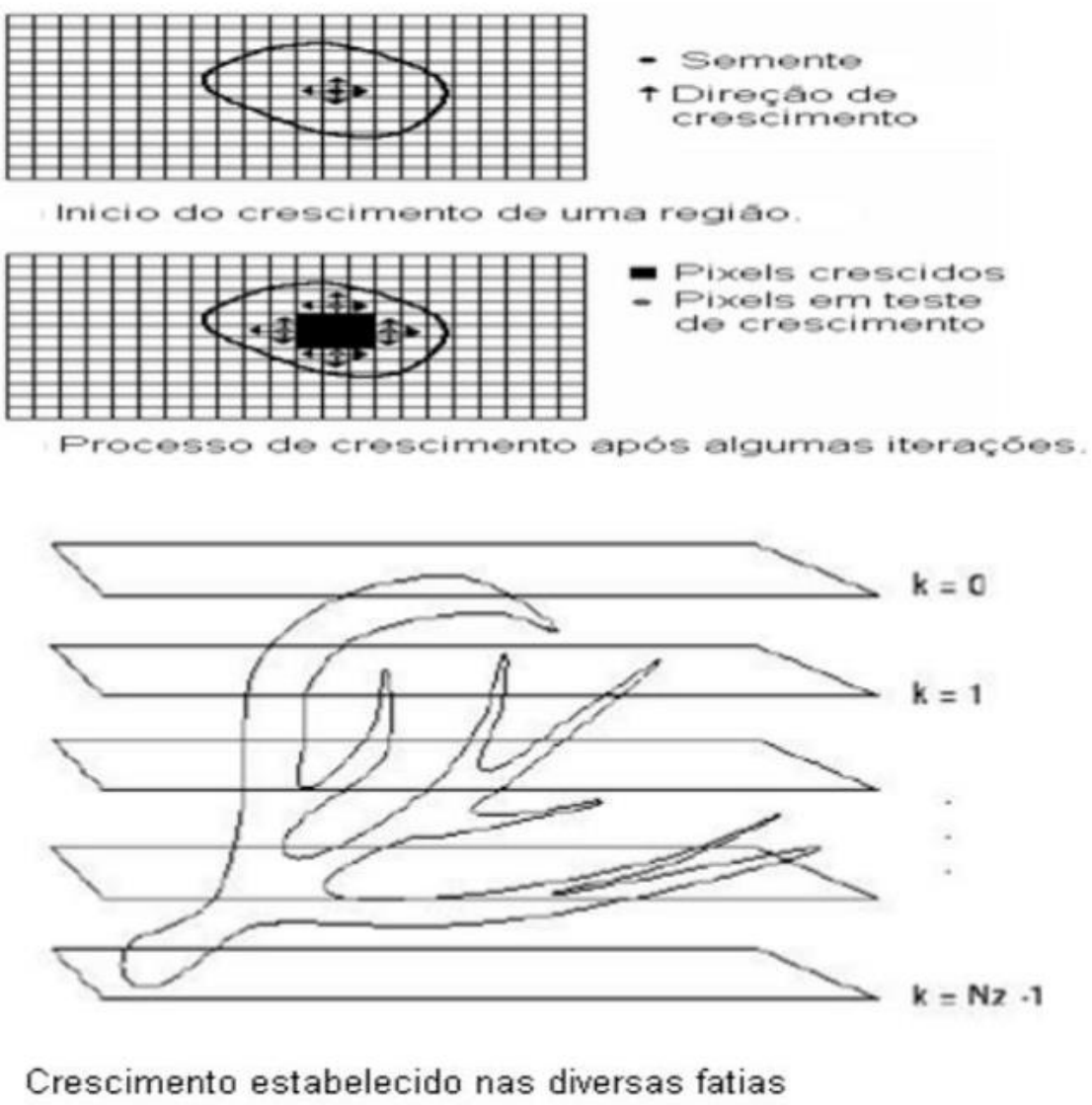


Figura 4.34. Crescimento de regiões 3D.

O algoritmo de crescimento de região (ACR) pode ser utilizado para resolver o problema dos clusters que possuírem em sua composição partes desconexas. Os parâmetros utilizados pelo ACR são: a semente de crescimento, o tamanho da vizinhança e a intensidade de cinza para essa vizinhança, ou seja, o crescimento é feito até que seja encontrado ao menos um vizinho que não possua a mesma intensidade da semente. Dessa forma o ACR irá realizar uma etapa simples de isolamento de todas as estruturas individualmente para uma análise posterior.

A Figura 4.35 mostra um exemplo de aplicação de um ACR em exames de tomografia computadorizada do tórax para detecção de nódulos pulmonares.

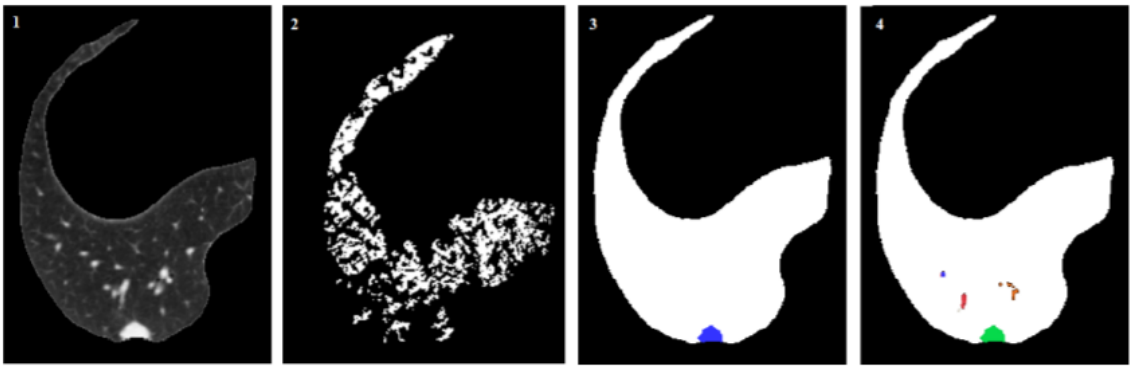


Figura 4.35. 1) Imagem original (2) Cluster grande isolado. (3) Cluster menor isolado (4) Resultado Final. Fonte: Adaptado de por [Carvalho Filho 2013].

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkConnectedThresholdImageFilter.h"
#include "itkCastImageFilter.h"
#include "vtkImageViewer.h"
#include "vtkRenderWindowInteractor.h"
#include "vtkSmartPointer.h"
#include "vtkImageActor.h"
#include "vtkInteractorStyleImage.h"
#include "vtkRenderer.h"

typedef itk::Image< PixelType, Dimension >  ImageType;

static void CreateImage(ImageType::Pointer image);

int main( int argc, char *argv[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  typedef itk::ConnectedThresholdImageFilter<ImageType, ImageType> ConnectedFilterType;
  ConnectedFilterType::Pointer connectedThreshold = ConnectedFilterType::New();
  float lower = 95.0;
  float upper = 105.0;
  connectedThreshold->SetLower(lower);
  connectedThreshold->SetUpper(upper);

  connectedThreshold->SetReplaceValue(255);

  // Seed 1: (25, 35)
  ImageType::IndexType seed1;
  seed1[0] = 25;
  seed1[1] = 35;
  connectedThreshold->SetSeed(seed1);
  connectedThreshold->SetInput(image);
```

```
// Seed 2: (110, 120)
ImageType::IndexType seed2;
seed2[0] = 110;
seed2[1] = 120;
connectedThreshold->SetSeed(seed2);
connectedThreshold->SetReplaceValue(150);

connectedThreshold->SetInput(image);

// Visualize
typedef itk::ImageToVTKImageFilter<ImageType> ConnectorType;
ConnectorType::Pointer connector2 = ConnectorType::New();
connector2->SetInput(image2);

vtkSmartPointer<vtkImageActor> actor2 =
  vtkSmartPointer<vtkImageActor>::New();
actor2->SetInput(connector2->GetOutput());

// Visualize joined image
ConnectorType::Pointer addConnector = ConnectorType::New();
addConnector->SetInput(addFilter->GetOutput());

vtkSmartPointer<vtkImageActor> addActor =
  vtkSmartPointer<vtkImageActor>::New();
addActor->SetInput(addConnector->GetOutput());

// There will be one render window
vtkSmartPointer<vtkRenderWindow> renderWindow =
  vtkSmartPointer<vtkRenderWindow>::New();
renderWindow->SetSize(900, 300);
vtkSmartPointer<vtkRenderWindowInteractor> interactor =
  vtkSmartPointer<vtkRenderWindowInteractor>::New();
interactor->SetRenderWindow(renderWindow);

// Define viewport ranges
// (xmin, ymin, xmax, ymax)
double leftViewport[4] = {0.0, 0.0, 0.33, 1.0};
double centerViewport[4] = {0.33, 0.0, 0.66, 1.0};
double rightViewport[4] = {0.66, 0.0, 1.0, 1.0};

// Setup both renderers
vtkSmartPointer<vtkRenderer> leftRenderer =
  vtkSmartPointer<vtkRenderer>::New();
renderWindow->AddRenderer(leftRenderer);
leftRenderer->SetViewport(leftViewport);
leftRenderer->SetBackground(.6, .5, .4);

vtkSmartPointer<vtkRenderer> centerRenderer =
  vtkSmartPointer<vtkRenderer>::New();
renderWindow->AddRenderer(centerRenderer);
centerRenderer->SetViewport(centerViewport);
centerRenderer->SetBackground(.4, .5, .6);

vtkSmartPointer<vtkRenderer> rightRenderer =
  vtkSmartPointer<vtkRenderer>::New();
renderWindow->AddRenderer(rightRenderer);
rightRenderer->SetViewport(rightViewport);
rightRenderer->SetBackground(.4, .5, .6);
```

```
  // Add the sphere to the left and the cube to the right
  leftRenderer->AddActor(actor1);
  centerRenderer->AddActor(actor2);
  rightRenderer->AddActor(addActor);
  leftRenderer->ResetCamera();
  centerRenderer->ResetCamera();
  rightRenderer->ResetCamera();
  renderWindow->Render();
  vtkSmartPointer<vtkInteractorStyleImage> style =
    vtkSmartPointer<vtkInteractorStyleImage>::New();
  interactor->SetInteractorStyle(style);

  interactor->Start();
  return EXIT_SUCCESS;
}

void CreateImage(ImageType::Pointer image)

  // Create an image with 2 connected components
  ImageType::RegionType region;
  ImageType::IndexType start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  size[0] = 200;
  size[1] = 300;

  region.SetSize( size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

     for(unsigned int c = 30; c < 100; c++)
       {
       ImageType::IndexType pixelIndex;
       pixelIndex[0] = r;
       pixelIndex[1] = c;

       image->SetPixel(pixelIndex, 100.0);
       }
     }

   // Make another square
   for(unsigned int r = 100; r < 130; r++)
     {
     for(unsigned int c = 115; c < 160; c++)
       {
       ImageType::IndexType pixelIndex;
       pixelIndex[0] = r;
       pixelIndex[1] = c;

       image->SetPixel(pixelIndex, 100.0);
       }
     }
}
```

Figura 4.36. Código fonte para método de crescimento de região com ConnectedThresholdImageFilter

4.6. Conclusão

O processamento digital de imagens fornece ferramentas que facilitam a identificação ou extração de informações pertinentes contidas em imagens digitais, para interpretações posteriores aos processos desencadeados. Dessa maneira, sistemas computacionais são utilizados em atividades interativas de análise e manipulação, tornando evidentes determinadas peculiaridades ou regiões de interesse nas imagens trabalhadas.

Através do processamento digital de imagens e a partir da problemática gerada por diversas doenças em nível mundial, a informática médica é constituída por sistemas computacionais para auxiliar o trabalho de especialistas e, consequentemente, a vida dos pacientes. Um dos aspectos da informática médica é a análise e manipulação de imagens relacionadas a diversos exames e doenças, de modo que a interpretação eficaz destas imagens é imprescindível para um diagnóstico consistente.

Os benefícios gerados pela informática médica são imensuráveis, pois reduzem possíveis erros humanos, automatizam determinados processos e/ou acrescentam informações pertinentes à detecção de doenças. Vale destacar que a informática médica perpassa desde os sistemas de auxílio ao diagnóstico a aplicativos que realizam a prevenção ou acompanhamento, tornando todos os esforços neste sentido significativos para a vida das pessoas.

Para a implementação destes sistemas, a biblioteca ITK mostrou-se a plataforma ideal. Sendo *software* livre, com o desenvolvimento colaborativo de soluções, ITK disponibiliza uma gama de ferramentas para a elaboração e implementação das mais diversas técnicas de processamento digital de imagens médicas. ITK possibilita a construção de sistemas completos de auxílio ao especialista, esmiuçar e compreender técnicas específicas de manipulação de imagens, ou construir novas abordagens e metodologias que contribuam para o contexto trabalhado.

Este trabalho apresentou a estrutura ITK, com seu conceito, aplicações e arquitetura e um conjunto de técnicas construídas pela comunidade, que perpassam por operações básicas (leitura e escrita de imagens e manipulação de matriz de *pixels*), ferramentas de pré-processamento (Filtros, detectores de borda, operadores morfológicos, lógicos e esqueletização) e métodos de segmentação por agrupamento (Clusterizadores) e limiarização. Através deste estudo e levantamento, pode-se construir sistemas completos da informática médica, para o tratamento e manipulação de imagens relacionadas aos mais variados tipos de doença, como câncer ou glaucoma, por exemplo.

4.7. Referências

Canny, J. (1986) “A Computational Approach to Edge Detection”

Carvalho Filho, A. O. (2013) Detecção Automática de Nódulos Pulmonares Solitários usando Quality Threshold Clustering e MVS. Dissertação de Mestrado na área de Ciência da Computação. (Programa de Pós-Graduação em Ciência da computação) - Universidade Federal do Maranhão, São Luís.

Carvalho Filho, A. O.; Silva, A. C.; Paiva, A. C.; Gattas, R. A. N. M. (2016) “Lung-nodule classification based on computed tomography using taxonomic diversity

indexes and an SVM". Journal of Signal Processing Systems for Signal, Image, and Video Technology, vol. 83.

Claro, M. L.; Araújo, F. H. D. (2015) "uso de classificadores para a detecção automática do glaucoma".

Filho, O.M; Neto, H.V (1999), "Processamento Digital de Imagens", Brasport, ISBN 8574520098, Rio de Janeiro.

Homann, H. (2007) "Implementation of a 3D thinning algorithm".

Insight Segmentation and Registration Toolkit -ITK (2016), Introdução ao Itk, https://itk.org/, setembro.

Lourenço, L. H. A. (2011) "paralelização do detector de bordas canny para a biblioteca itk utilizando cuda"

NETTO, S. M. B. (2010) Segmentação Automática de Nódulo Pulmonar com Growing Neural Gás e Máquina de Vetores de Suporte. Dissertação de Mestrado na área de Ciência da Computação. (Programa de Pós-Graduação em Engenharia de Eletricidade) - Universidade Federal do Maranhão, São Luís.

Oliveira, L. F. O.; Zanchet, B. A.; Barros, R. C.; Gomes, V. V.; Fujii, Y.; Vortmann, C. H.; Patzer, G; "Utilização das bibliotecas livres VTK e ITK no processamento de imagens médicas"

Sampaio, W. B. (2015) "detecção de massas em imagens mamografias usando uma metodologia adaptada à densidade da mama". Dissertação de mestrado (programa de pós-graduação em engenharia de eletricidade) – universidade federal do maranhão, São Luís.

Silva, A.M. (2001) "Curso Processamento digital de imagens de satélite", Centro de Eventos da PUCRS, Porto Alegre – RS.

Spring: Integrating remote sensing and GIS by object-oriented data modelling. (1996) "Camara G, Souza RCM, Freitas UM, Garrido J" Computers & Graphics, 20: (3) 395-403.

Yoo, S. T.; Ackerman, M. J.; Lorensen, W. E.; Schoroeder, W; Chalana, V; Aylward, S.; Metaxas, D.; Whitaker, R; (2002) "Engineering and Algorithm Design for na Image Processing API: A Technical Report on ITK - the Insight Toolkit" Studies in Health Technology and Informatics, vol. 85 (Proceedings of Medicine Meets Virtual Reality 02/10. J. D. Westwood, et al., eds.),

Ziemer, P. G. P.; Collares, M.; Camargo, E.; Freitas, I. C.; Blanco, P. J.; Feijóo, R. A.; "ImageLab: Um sistema Multi-Orientado na Visualização e Processamento de Imagens Médicas" .

Uma proposta de mapeamento da *Li-Fraumeni Ontology* com a BioTop Lite

Lhicyara Allaynne Estevam Avelino[1]
Ph.D Ricardo Moura Sekeff Budaruiche[2]

Resumo: O objetivo deste estudo é mostrar uma proposta de mapeamento da *Li-Fraumeni Ontology* com uma *Upper Level Ontology* (ULO), identificando os pontos de conexão entre as duas com a finalidade de mostrar, por meio de inferência, sua validação. O mapeamento será feito por meio de associação entre elementos semelhantes, em cada ontologia. A *Li-Fraumeni Ontology* é formada por 3 módulos: GenOnto, CDOnto e LFOnto, foi desenvolvida para classificar pacientes de uma família segundo os critérios da síndrome. Estes módulos serão mapeados com a ontologia de alto nível BioTop Lite, uma ontologia de biomedicina que foi lançada usando lógica descritiva e OWL-DL. O resultado esperado deste estudo é apresentar, como resultado do mapeamento, a validade dos termos e conceitos da *Li-Fraumeni Ontology*.

Palavras-chave: BioTop Lite. *Li-Fraumeni Ontology*. Mapeamento de ontologias.

Abstract: *The objective of this study is to show a proposed mapping of Li-Fraumeni Ontology with an Upper Level Ontologies (ULO), identifying the connection points between the two in order to show by inference, its validation. The mapping will be done associating similar elements in each ontology. The Li-Fraumeni Ontology is composed by three modules: GenOnto, CDOnto and LFOnto. It was developed to help classifying patients accordingly to a family syndrome criteria. These modules will be mapped to a high level ontology called BioTopLite, a biomedical ontology that was launched using description logic and OWL-DL. The expected result of this study is to present, as result of mapping, a validity of the terms and concepts of Li-Fraumeni Ontology.*

Keywords: BioTop Lite. Li-Fraumeni Ontology. Ontology mapping.

1 Introdução

Nos últimos anos, o uso e a construção de ontologias tem ganhado destaque tanto por ser um meio de organizar o conteúdo de fontes de dados, como no uso em sistemas inteligentes. Fato que se justifica pela crescente na quantidade do número de dados gerados por pessoas e por sistemas. Uma das definições mais conhecidas para ontologias é dada por Gruber [1] "Uma ontologia é uma especificação explícita de uma conceitualização. [...]". Elas podem servir para diversos propósitos e tem como principal objetivo descrever uma maneira estruturada de organizar o que se sabe sobre algum domínio do conhecimento para a utilização na construção das bases de conhecimento.

As ontologias tem sua classificação definida em: *Upper Level Ontologies* (ULO), Ontologias de Domínio, Ontologias de Tarefas, Ontologias de Aplicação e Ontologias de Representação. Essa classificação depende da função que elas desempenham no domínio do conhecimento em questão [2]. No caso das ontologias de alto nível podemos dizer que modelam o senso comum para estabelecer uma semântica de conceitos comum a ser utilizada por outras ontologias; seu uso permite verificar se duas ou mais ontologias de domínio são semanticamente equivalentes no que diz respeito aos conceitos, propriedades e axiomatizações, sendo esse um dos maiores benefícios para utilizá-las durante a construção de ontologias de domínio: prover "interoperabilidade semântica" para as ontologias que possuem domínios cruzados.

Entendemos por mapeamento de ontologia a tarefa de associar os elementos com alguma relação semântica em cada ontologia superando, desta maneira, desvantagens que são causadas quando se possui

[1] Bacharel em Ciência da Computação
{lhicyarae@gmail.com}
[2] Faculdade ESTÁCIO-CEUT
{sekeff@usp.br}

somente uma ontologia e por falta de capacidade na interpretação semântica. A BioTop Lite é uma ontologia de alto nível no domínio da biomedicina e foi descrita usando lógica de descrição e OWL-DL. É bastante utilizada no desenvolvimento de outras ontologias e vem recebendo melhorias, sendo uma camada de alto nível com desempenho consideravelmente melhor que as demais. [3]

Dessa forma, pretende-se mostrar uma proposta de mapeamento da *Li-Fraumeni Ontology*, uma ontologia para a Síndrome de Li-Fraumeni desenvolvida para classificar pacientes de uma família segundo os critérios da síndrome, com uma ontologia de nível superior, a BioTop Lite, uma ontologia para representar conhecimento no domínio da ciência biológica, identificando os pontos de conexão entre as duas. O intuito é mostrar, por meio de inferência, a validade semântica dos termos da *Li-Fraumeni Ontology*.

2 Ontologias

Atualmente, o aumento exponencial de dados tem dificultado a busca por informações. Diante disso, a organização dessas informações tem sido de suma importância, levando à busca por técnicas de melhorias no tratamento de dados e na representação de conhecimento, onde as ontologias tem ganhado destaque como um meio de organizar o conteúdo dessas fontes de dados.

Para a Ciência da Computação existem inúmeras definições. Uma definição que se mostra completa e de certa forma simples, seria, "Uma ontologia é uma especificação formal e explicita de uma conceitualização compartilhada" [4]. A "formalidade" citada nesta definição seria a legibilidade otimizada para computadores; conceitos, relações, propriedades, axiomas, funções e restrições seria a "especificação"; "compartilhado" refere-se a conhecimento consensual; e "conceitualização" seria um modelo abstrato referente a algo no mundo real.[5]

O uso e a construção de ontologias em sistemas inteligentes é justificado pelo aumento no número de dados gerados tanto por pessoas quanto por sistemas, assim como por organizações que perceberam a necessidade em estruturar suas informações buscando conhecimento. Para esta estruturação são criados critérios e interesses, gerando conhecimentos diversos sobre os mesmos domínios, que ao serem vistos por uma ótica comum, possibilitam que os responsáveis tomem decisões diferentes. As ontologias podem servir a diversos propósitos diferentes, além da representação do conhecimento. Nessa situação, elas podem ser usadas para integrar sistemas diferentes, dentro e fora de uma mesma organização, por meio da uma camada intermediária que mapeia ou traduz conceitos diferentes sobre coisas e objetos de um mesmo domínio.

Segundo Noy e McGuinness [6], todas as ontologias são formadas pelos mesmos elementos: classes, relações, atributos, axiomas e indivíduos. As classes descrevem entidades com características comuns. As relações representam as restrições semânticas existentes entre os indivíduos de duas classes. Estas restrições estabelecem, não de maneira isolada, as bases semânticas de uma ontologia, na medida em que apresentam axiomas que limitam a classificação de indivíduos dentro das classes, de acordo com o modelo conceitual. Também é possível que uma classe seja subclasse de outra. Os atributos representam os dados específicos sobre determinado indivíduo que pertence a determinada classe. Esses atributos são, na realidade, as características que identificam esse indivíduo como sendo parte da classe em que ele se encontra. O indivíduo, por sua vez, representa uma instanciação de um representante de uma determinada classe.

As ontologias são classificadas em cinco tipos, dependendo da função que elas desempenham no domínio em questão [7][8]:

• Ontologias *Upper-Level* (ULO): descrevem, de maneira genérica, conceitos que são independentes do domínio do problema. São uma tentativa de estabelecer os fundamentos, um consenso entre as partes envolvidas, à respeito dos conceitos descritos por elas [9].

• Ontologias de Domínio: descrevem "mini-mundos" específicos do domínio do problema, como conceitos sobre biomedicina ou vinhos. É comum as ontologias de domínio fazerem uso de conceitos e relações definidas nas ontologias *top-level*;

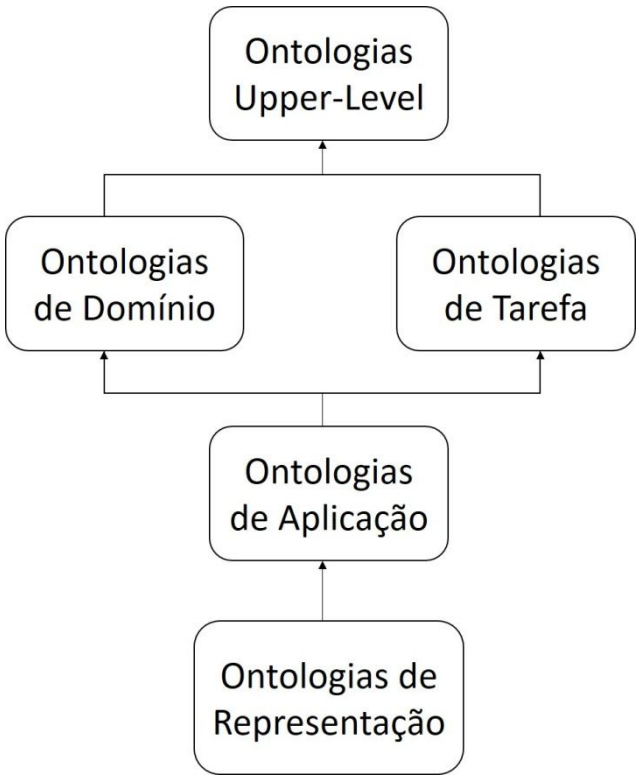
• Ontologias de Tarefa: utilizadas para descrever tarefas, processos ou atividades independentemente da situação-problema, como, por exemplo, o diagnóstico de determinada doença, o processo de fabricação de vinhos ou etapas da construção de um veículo. As ontologias de tarefa, assim como as de domínio, também fazem uso dos conceitos e relações estabelecidos pelas ontologias *Upper-Level*;

• Ontologias de Aplicação: Descrevem conceitos que dependem tanto de uma ontologia de domínio particular quanto de uma ontologia de tarefa específica. Estes conceitos normalmente correspondem a regras aplicadas a entidades de domínio enquanto executam determinada tarefa [8].

• Ontologias de Representação: explicam as conceituações que fundamentam os formalismos de representação de conhecimento, procurando tornar claros os compromissos ontológicos embutidos nestes formalismos [9].

Essa classificação sugere uma dependência funcional entre as ontologias, onde as que estão nos níveis mais inferiores, dependem dos conceitos e relações estabelecidos por aquelas de níveis superiores, conforme a figura 1.

Figura 1: Dependência Funcional entre Ontologias



3 Ontologias de nível superior (ULO)

Também conhecidas por Ontologias *top-level* ou Ontologias Fundamentais, descrevem, de maneira genérica, conceitos que são independentes do domínio do problema. São uma tentativa de estabelecer os fundamentos, um consenso entre as partes envolvidas, à respeito dos conceitos descritos por elas [10]. Diferentes ontologias de nível superior oferecem diferentes distinções com base nos tipos de entidades que incluem, suas teorias de espaço e tempo, bem como a relação dos indivíduos com o espaço e o tempo.

Pode-se dizer que essas ontologias modelam o senso comum para estabelecer uma semântica de conceitos comum a ser utilizada por outras ontologias; O uso das Ontologias de Nível Superior permite a relação semântica existente entre dois ou mais termos de uma ontologia com a ontologia de nível superior. Sendo esse um dos maiores benefícios do seu uso durante a construção de ontologias de domínio: prover "interoperabilidade semântica" para as ontologias que possuem domínios cruzados. O fato dessas ontologias estarem organizadas em taxonomias permite estabelecer uma relação explicita entre as categorias de uma Ontologias de Nível Superior e apenas algumas categorias, as mais superiores na hierarquia, das ontologias de domínio [10].

Através de axiomas as *Upper-Level* oferecem restrições sobre as categorias que eles fornecem. Estas restrições são herdadas pelas ontologias de domínio que são fundadas nas ontologias de nível superior. Consequentemente, ontologias de nível superior fornecem um meio para verificar ontologias de domínio de forma que a ULO agrupa as categorias de relações hierárquicas da forma mais genérica e abstrata possível e atribui a elas propriedades que as distinguem umas das outras. Esta particularidade é útil quando uma nova ontologia é desenvolvida com a intenção de interagir semanticamente com uma ontologia já existente. Além disso, eles podem fornecer uma compatibilidade de alto nível e verificação de plausibilidade para ontologias de domínio e sua integração semântica [10]. São exemplos de ontologias *top-level*: *Basic Formal Ontology* (BFO), *Descriptive Ontology for Linguistic and Cognitive Engineering* (DOLCE), *General Formal Ontology* (GFO), *Suggested Upper Merged Ontology* (SUMO), *KR Ontology*, *Cycupper ontology* e BioTop Lite.

4 BioTop Lite

BioTop Lite é uma ontologia *Upper-Level* de biomedicina e sua criação foi motivada por problemas de desempenho graves relacionados a sua antecessora BioTop. Esta ontologia possui classes, relações e axiomas necessários para suprir às necessidades da maioria das ontologias da ciência da vida. Baseada em sua antecessora, a BioTop Lite foi lançada usando lógica de descrição e OWL-DL, a mesma não se destina a competir com outras ULO's, mas, possibilita se integrar. Desta maneira, ela pode ser empregada numa camada de alto nível sem restringir outra ULO. Tem forte foco em axiomas, sendo um mecanismo importante para verificação de consistência.

Uma decisão importante do projeto BioTop e BioTop Lite aborda a inerente ambiguidade de termos médicos: "*Fracture*" pode designar tanto um processo de quebra ou fratura, bem como o seu resultado, um osso fraturado. "*Allergy*" pode ser interpretado como uma predisposição alérgica ou como manifestação alérgica. [3] Tais distinções categoriais, muitas vezes, não se refletem nem no discurso e no modo de pensar dos médicos nem nas terminologias médicas, e para muitos padrões de raciocínio clínico uma distinção não é necessária.

Um problema na BioTop Lite é a representação das relações entre entidades continuas, ou seja, objetos que existem durante o tempo, sofrer alteração temporal e não ter partes temporais como processos ou intervalos de tempo. A linguagem OWL-DL não conta para a representação de tempo, gerando graves consequências. Contudo, esta ULO é bastante utilizada no desenvolvimento de outras ontologias e vem recebendo melhorias, sendo uma camada de alto nivel com desempenho consideravelmente melhor que as demais [3].

5 *Li-Fraumeni Ontology*

A síndrome de Li-Fraumeni é uma doença rara de predisposição ao câncer de alta penetrância, que apresenta caráter autossômico dominante. Está ligada a mutações do gene TP53, que normalmente ajuda no controle do ciclo celular. Estima-se que pacientes com SLF e sua variante Li-Fraumeni Lite apresentam 50% de chance de desenvolver tumores antes dos 40 anos de idade, comparados a 1% na população geral, e que 90% dos portadores desenvolvam câncer até 60 anos de idade.

A caracterização inicial do espectro tumoral da síndrome incluía como critérios principais para o diagnóstico a presença de osteossarcomas, sarcomas de partes moles, câncer de mama em mulheres na pré-menopausa, tumores cerebrais, tumores adrenocorticais e leucemias agudas sendo inicialmente denominada SBLA (*sarcoma, breast, leukemia and adrenocortical tumor syndrome*). No entanto, desde a descrição inicial da síndrome, verificou-se um amplo espectro tumoral, incluindo tumores primários malignos de estômago, cólon, pâncreas, esôfago, além de tumores de células germinativas e melanoma.[11]

Para se classificar pacientes portadores da síndrome, existem quatro critérios clínicos, onde um paciente pode se enquadrar em um ou mais desses critérios [12]. São eles:

• Critério Clássico: São os critérios clássicos definidos por Li e Fraumeni para avaliação clínica de pacientes portadores da Síndrome de *Li-Fraumeni*: (1) paciente que tenha apresentado algum tipo de sarcoma antes dos 45 anos, (2) um parente de primeiro grau com qualquer tipo de câncer antes dos 45 anos e (3) um segundo parente próximo (primeiro ou segundo grau) que também tenha apresentado ou um sarcoma, não importando a idade, ou qualquer tipo de câncer antes dos 45 anos.

• Critério Chompret: o primeiro conjunto de critérios clínicos definidos como Chompret foram estudados procurando contornar as especificidades do critério Clássico. Depois das revisões que passou por meio de pesquisas com pacientes, foi definido da seguinte forma: (1) Tumor pertencente ao espectro de tumores LFS - sarcoma de tecido mole, osteosarcoma, câncer da mama pré-menopausa, tumor cerebral, carcinoma adrenocortical, leucemia ou câncer bronco-pulmonar - antes de 46 anos de idade e pelo menos um parente de primeiro ou segundo grau com um tumor do espectro de tumores LFS (exceto câncer de mama se o probando tem câncer de mama) antes de 56 anos de idade ou qualquer parente do probando com múltiplos tumores primários em qualquer idade; ou (2) Um probando com vários tumores (exceto múltiplos tumores mamários), dois deles pertencentes ao espectro de tumores LFS e o primeiro ocorrido antes dos 46 anos de idade; ou (3) Um probando que é diagnosticado com carcinoma adrenocortical ou tumor do plexo coroide, independente da história familiar;

• Critério Eeles: também é conhecido por ser um critério mais relaxado e incluir pacientes que não apresentam todas as características dos critérios Clássicos mas que podem sugerir a presença da mutação no TP53. Nesse critério o que importa são os tipos de câncer desenvolvidos nos seus parentes de primeiro ou segundo grau: (1) Dois parentes de primeiro ou segundo grau com tumores típicos de LFS (sarcoma, leucemia ou câncer de mama, cérebro ou córtex adrenal) em qualquer idade;

• Critério Birch: apresenta a presença de rabdomiossarcomas e carcinomas adrenocortical, apesar de não serem descritos nos critérios Clássicos, mas que representam fatores na presença de mutações no gene TP53: (1) Um probando com qualquer tipo de câncer na infância ou sarcoma, tumor cerebral ou carcinoma adrenocortical diagnosticados antes dos 45 anos; (2) Um parente de primeiro grau ou segundo grau com um tumor maligno típico de LFS (sarcoma, leucemia ou câncer de mama, cérebro ou córtex adrenal), independente da idade no momento do diagnóstico; (3) Um parente de primeiro grau ou segundo grau com qualquer câncer diagnosticado antes dos 60 anos.

A ontologia de Li-Fraumeni é utilizada para representar o conhecimento para a classificação de famílias que reúnem critérios clínicos para a síndrome de Li-Fraumeni. Foi construída de forma modular pretendendo facilitar o projeto no que se refere à manutenção e correções da ontologia, além de facilitar o reuso da base de conhecimento, além de também ser um projeto de desenvolvimento colaborativo, pois os especialistas de domínio, os engenheiros do conhecimento e o construtor da ontologia estavam geograficamente distantes [12]. A *Li-Fraumeni Ontology* é formada por três ontologias desenvolvidas como módulos: *Genealogy Ontology, Clinical Data Ontology* e a *Li-Fraumeni Ontology*.

5.1 Genealogy Ontology - GenOnto

É uma ontologia de referência desenvolvida para a *Li-Fraumeni Ontology* seguindo um padrão baseado em recomendações da literatura genética, onde o grau de parentesco é relacionado ao número de genes compartilhados pelos indivíduos. Seu propósito é representar as relações familiares capazes de proporcionar o compartilhamento genético entre os indivíduos necessários para a identificação da síndrome. A GenOnto possui apenas a classe **Person**, esta representa todos os pacientes e familiares, tornando subclasses desnecessárias na ontologia. Os relacionamentos familiares descritos são: *hasPartner*, *hasChild*, *hasSibling*, *hasCousin* e *hasUncles*, que serão os relacionamentos relevantes para a classificação dos pacientes; e propriedade do objeto que representam os graus de parentesco: *has3rdDgRelatives* e *hasSome1stOr2ndRelatives*.

5.2 Clinical Data Ontology – CDOnto

Foi desenvolvida com base em outra ontologia de referência, a SNOMED-CT, que modela o sistema de codificação ICD-10, uma estrutura de conceitos e metadados relacionados à Classificação Internacional de Doenças e Problemas Relacionados à Saúde (CID) versão 10, CID-10, e também contém a estrutura de classes da Classificação Internacional de Doenças para Oncologia, chamada CID-O. Esses dois sistemas de codificação de doenças utilizados no cenário médico, CID-10 e CID-O, foram representados como classes na ontologia, onde cada uma das subclasses que descrevem um código CID-10 ou CID-O é representada por um indivíduo do código em questão. Além das classes dos códigos acima citados, a ontologia apresenta uma classe **Document**, nela há a subclasse **Cancer_Diagnostic**, esta subclasse descreve que qualquer indivíduo classificado como documento que se relacione por meio da propriedade *hasDiagnosticCode* com qualquer indivíduo da classe CID10 ou CID-O será classificado como um **Cancer_Diagnostic**. O objetivo principal dessa ontologia é apresentar uma descrição do quadro clínico do paciente.

5.3 *Li-Fraumeni Ontology* – LFOnto

Desenvolvida usando os conceitos existentes até agora da Síndrome de Li-Fraumeni, assim como também as regras que definem os quatro critérios clínicos de diagnóstico da síndrome: Classic, Chompret, Eeles e Birch. O seu principal objetivo é prover axiomas que classifiquem automaticamente os pacientes em um ou mais critérios clínicos da síndrome. Na LFOnto existem três classes necessárias para a classificação dos pacientes: a *Birch_Spectrum_Diagnosis, Eeles_Spectrum_Diagnosis* e *Chompret_Spectrum_Diagnosis*. Nelas encontramos os critérios necessários para a classificação de cada critério da síndrome baseados na literatura.

6 Trabalhos Relacionados

Ian Niles [13] desenvolveu um projeto que estuda o mapeamento da WordNet à ontologia SUMO. WordNet é um banco de dados lexical projetado para o inglês, agrupando palavras ingleses em conjuntos de sinônimos chamados *synsets* onde fornece tanto definições como exemplos de uso e é utilizado principalmente na análise automática de texto e aplicações de inteligência artificial. Já a SUMO (*Suggested Upper Merged Ontology*) é uma ontologia de nível superior que descreve conceitos de meta-nível, entidades gerais que não pertencem a um domínio específico, destina-se como uma ontologia base para uma variedade de sistemas de processamento de informação do computador.

Por conta do extenso número de conceitos presentes na WordNet, foi decidido restringir inicialmente os termos que seriam mapeados, resultando em apenas synsets substantivo, pelo fato de que cada conceito SUMO tem a forma de um substantivo e pelo fato de que substantivos synsets em WordNet geralmente suportam mais as relações e contêm mais informações do que outros tipos de synsets. Para o mapeamento proposto no trabalho, foram usadas três possível relações de interesse: sinonímia, hiperonímia e instanciação. [13]

Estes mapeamentos WordNet/SUMO, continua Ian, irão ajudar a uma ontologia formal ser usada eficazmente por quem não possui formação em lógica e matemática; assim como ser usada automaticamente por aplicativos que processam texto livre; e em saber quando uma ontologia está completa. Ou seja, que estes mapeamentos podem funcionar como um índice de linguagem natural para os conceitos na ontologia, como uma ponte entre estes conceitos estruturados e texto livre, processado por um número cada vez maior de aplicações, e como uma "verificação de integridade" sobre o conteúdo da ontologia.

Reed e Lenat fizeram um estudo sobre mapeamentos com a Cyc [14]. A Cyc é um projeto de inteligência artificial que tem o intuito de desenvolver uma ontologia e base de conhecimento que abranja conceitos de senso comum utilizados no dia-a-dia. No trabalho feito por Stephen e Lenat foram estudados mapeamento da Cyc com a FIPS 14 (*Federal Information Processing Standards*), que são um conjunto de padrões que descrevem os códigos de processamento de documentos; a *Medical Subject Headings* (MeSH) a partir da versão expandida de 1997, da *National Library of Thesaurus* Medicina da *Medical Subject Headings*; a SENSUS, uma grande ontologia com 70.000 termos derivados da WordNet e do Alto Modelo Pennman, neste foram mapeamos 201 Termos SENSUS que combinavam termos Cyc existentes; com o *Open Directory*, ligando mais de dez mil termos de classe para Cyc e criando novos termos Cyc; a WordNet, onde foram mapeados mais de 12.000 termos Cyc para WordNet versão 1.6; entre outros projetos listados e comentados no estudo.

Newton Miyoshi e Joaquim Felipe abordam uma proposta de trabalho para a construção de uma ontologia para o Registro de Evolução Clínica (REC) de um paciente, baseados no guia 101 de Noy e McGuiness, tendo como base a Rede Semântica da UMLS (*Unified Medical Language System*) e a estruturação da Informação do REC. A ontologia construída é especializada nos seguintes domínios: dermatologia, oncopediatria e gastroenterologia cirúrgica [15]. Miyoshi e Felipe refletem a dificuldade em relacionar dados biomédicos por conta da complexidade e heterogeneidade entre os sistemas, problema contornado com a utilização da Rede Semântica da UMLS como uma ontologia de alto-nível.

7 Metodologia

Neste estudo foi utilizado o método de abordagem com base na lógica dedutiva, pois a dedução é o caminho das consequências, é uma cadeia de raciocínio em conexão descendente, isto é, do geral para o particular, leva à conclusão. Segundo esse método, partindo-se de teorias e leis gerais, pode-se chegar à determinação ou previsão de fenômenos particulares [16]

Quanto aos objetivos esta pesquisa pode ser caracterizada como explicativa. Entende-se como pesquisas explicativas aquelas que têm como preocupação central identificar os fatores que determinam ou que contribuem para a ocorrência dos fenômenos [17]. Dessa forma, a pesquisa mapeou conceitos e termos da *Li-Fraumeni Ontology* com uma ontologia de nível superior, a BioTop Lite, identificando os pontos de conexão entre as duas e mostrando que a *Li-Fraumeni Ontology* é válida para aquele domínio.

De acordo com os procedimentos técnicos que foram utilizados, o modelo traçado para esta pesquisa é o bibliográfico experimental, pois a pesquisa bibliográfica é desenvolvida com base em material já elaborado, constituído principalmente de livros e artigos científicos e a pesquisa experimental é quando se pretende determinar um objeto de estudo, selecionar as variáveis que seriam capazes de influenciá-lo e definir as formas de controle e observação dos efeitos que a variável produz no objeto [17].

O processo ocorreu da seguinte forma: foi estudado, por meio de pesquisas em artigos, os três módulos da *Li-Fraumeni Ontology* e como ela está organizada, da mesma forma foi feita com a BioTop Lite; o passo seguinte foi encontrar os pontos de junção entre estas ontologias e descrever o mapeamento entre elas; por fim, foi implementado e mostrado, com este processo, por meio de inferência, que a ontologia é válida.

A classificação quanto ao processo do estudo será qualitativa, pois considera-se que existe uma relação entre o mundo e o sujeito que não pode ser traduzida em números. O que justifica esta classificação é que o resultado do teste não se dará de forma quantitativa, não será expresso em números, e sim a validação da ontologia partindo do mapeamento proposto.

8 Métodos

O processo de mapeamento dos três módulos da ontologia de Li-Fraumeni com a BioTop Lite ocorreu de forma separada com o intuito de manter o modelo de modularização pensado para o seu desenvolvimento. Além disso, as ontologias estudadas permitem que apenas as suas categorias mais relevantes sejam relacionadas com as categorias da ULO. Na CDOnto, apesar do tamanho extenso, o mapeamento ocorreu apenas com três superclasses da classe principal **Thing**: **Classification**, **Person** e **Document**. **Classification** se refere a alguns códigos do CID relacionados ao câncer, esta classe se adequou à descrição da classe '**information object**' da BioTop Lite, pois esta refere-se à uma informação, não necessariamente humana, uma vez que existe independente de qualquer portador material, veja a Figura 2. A classe **Person** foi relacionada à classe **organism** da ULO, que tem como definição estruturas biológicas autônomas, veja a Figura 3. Já a classe **Document** foi relacionada à **quality**, pois esta se refere à qualidade de uma entidade. Veja a Figura 4, produzindo a hierarquia de classes mostrada na Figura 5.

Figura 2: Representação do mapeamento entre a classe Classification, pertencente a *Li-Fraumeni Ontology*, com a classe 'information object', pertencente a BioTop Lite.

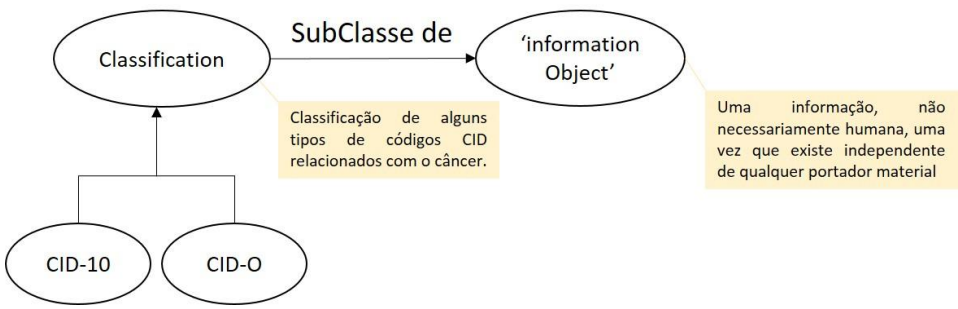


Figura 3: Representação do mapeamento entre a classe Person, pertencente a *Li-Fraumeni Ontology*, com a classe organism, pertencente a BioTop Lite.

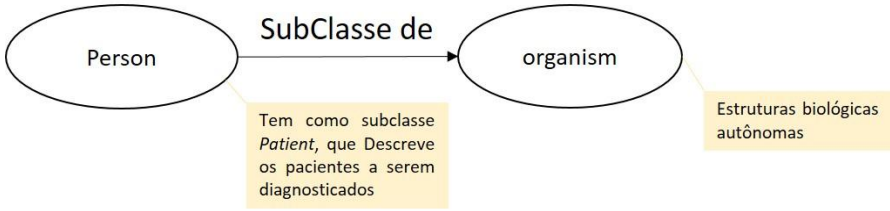


Figura 4: Representação do mapeamento entre a classe Document, pertencente a *Li-Fraumeni Ontology*, com a classe quality, pertencente a BioTop Lite.

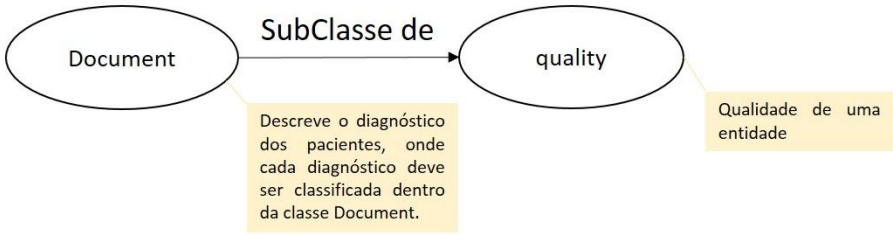
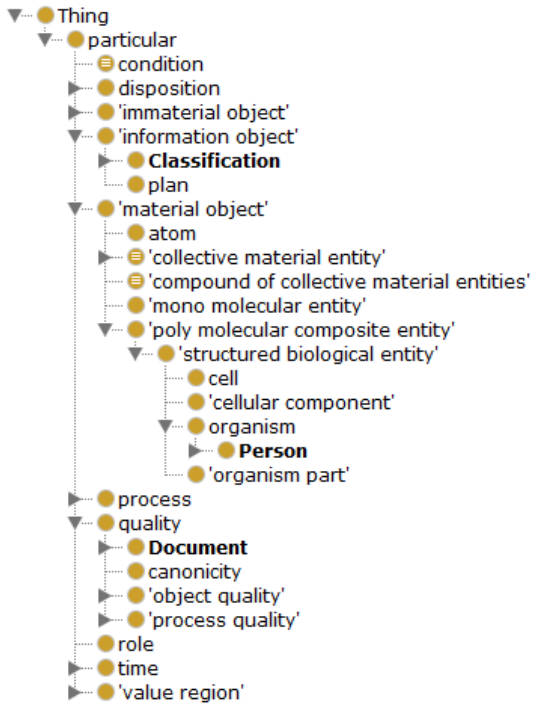


Figura 5: Mapeamento das classes da CDOnto.



As propriedades mapeadas da CDOnto foram três: *hasDiagnosticCode* foi relacionada como subpropriedade de '*abstractly related to'*, pois nesta é onde estão envolvidas entidades imateriais (entidades de informações, papéis, qualidades), veja a Figura 6. Por fim, as propriedades *hasDiagnosisDocument* e sua propriedade inversa *isFrom*, no mapeamento, estão como subpropriedades de '*causally related to*', pois esta manifesta uma cadeia causal entre entidades materiais ou processuais causalmente relacionadas, veja a Figura 7, resultando na hierarquia mostrada na Figura 8.

Figura 6: Representação do mapeamento entre a propriedade hasDiagnosticCode, pertencente a *Li-Fraumeni Ontology*, com a propriedade 'abstractly related to', pertencente a BioTop Lite.

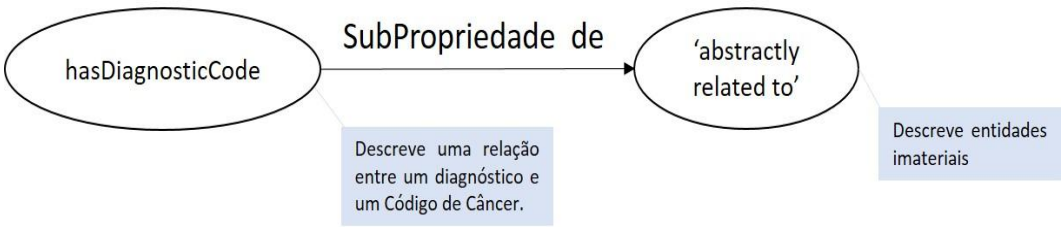


Figura 7: Representação do mapeamento entre a propriedade hasDiagnosisDocumente e sua propriedade inversa isFrom, pertencentes a *Li-Fraumeni Ontology*, com a propriedade 'causaly related to', pertencente a BioTop Lite.

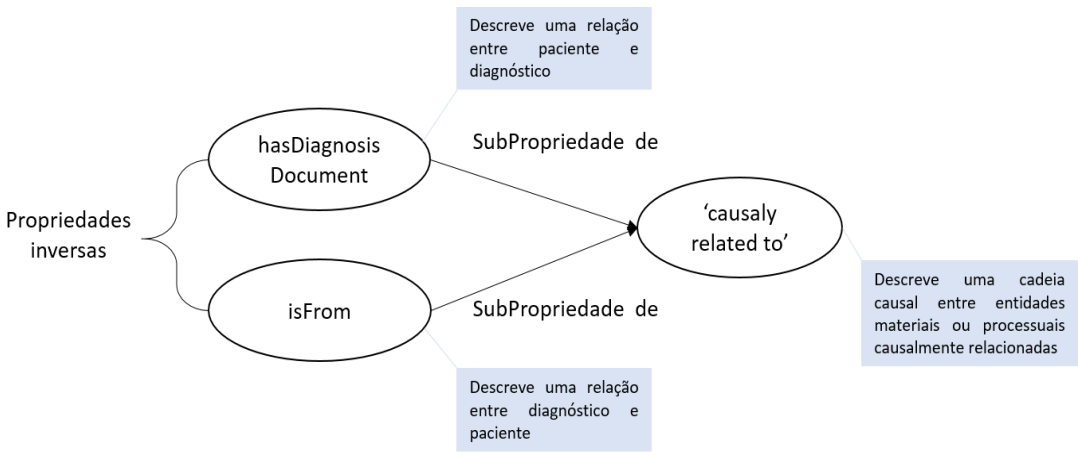
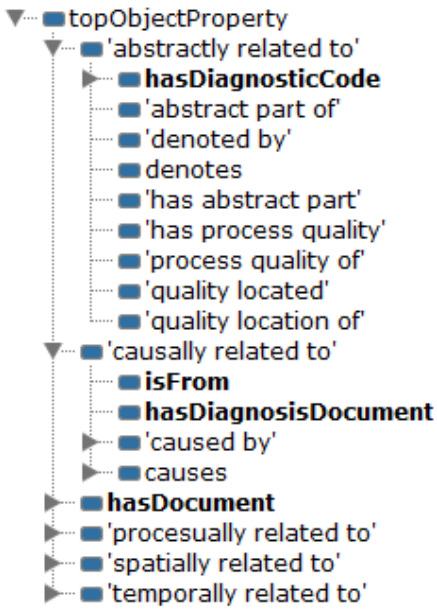


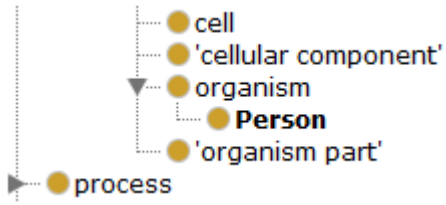
Figura 8: Mapeamento das propriedades da CDOnto



A ontologia GenOnto, como mencionado anteriormente, possui apenas a classe **Person**, dessa forma, o cruzamento ocorreu entre **Person** e a classe **organism** da BioTo Lite, semelhante a ocorrida no processo de

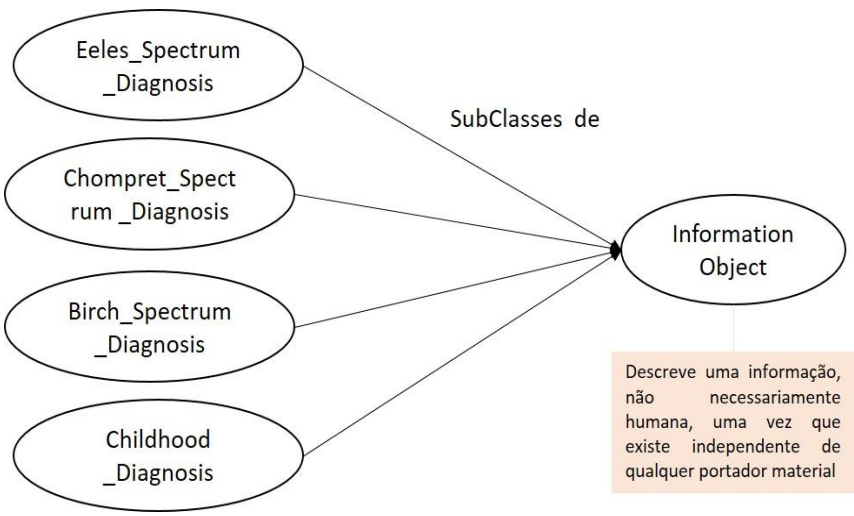
mapeamento da CDOnto, e nenhuma de suas propriedades se enquadrou nas propriedades existente da ULO, veja a Figura 9.

Figura 9: Mapeamento das classes da GenOnto



Na LFOnto foram mapeadas várias subclasses à apenas quatro classes da BioTop Lite. As subclasses que representavam críteros para o diagnóstico de classificação da sindrome (Birch_Spectrum_Diagnosis, Eeles_Spectrum_Diagnosis e Chompret_Spectrum_Diagnosis) também foram mapeadas como subclasses da classe 'information object' presente na BioTop Lite, veja a Figura 10. Já as classes que representam os tipos de câncer descritos na LFOnto foram mapeadas como subclasses da classe disposition presente na *Upper Level Ontology*, esta tem como definição "uma entidade de realização, veja a Figura 11. Sua manifestação é um processo, seu portador está envolvido em virtude da composição física do portador". A classe Person também está mapeada como subclasse de organism, assim como demonstrado no mapeamento realizado com a CDOnto, e DiagsLess60 como subclasse de 'point in time', veja a Figura 12, resultando na hierarquia de classes mostrada na Figura 13.

Figura 10: Representação do mapeamento entre as classes que descrevem os critérios clínicos da síndrome, pertencentes a *Li-Fraumeni Ontology*, com a classe InformationObject, pertencente a BioTop Lite.



Quanto às suas propriedade, a propriedade *hasDiagnosticCode* foi mapeada como equivalente a propriedade '*quality location of*' e como subpropriedade de '*abstractly related to*', que descreve relações em que estão envolvidas entidades imateriais (entidades de informações, papéis, qualidades), veja a Figura 14. abstratamente relacionadas e *hasDocument* está como subpropriedade de '*has patient*', esta última propriedade é inversa de *'patient in'* e relata um participante com um processo, com a condição de que este participante não é causalmente ativo, veja a Figura 15. A hierarquia de propriedades resultantes está demonstrada na Figura 16.

Figura 11: Representação do mapeamento entre as classes que descrevem os tipos de câncer, pertencentes a *Li-Fraumeni Ontology*, com a classe Disposition, pertencente a BioTop Lite.

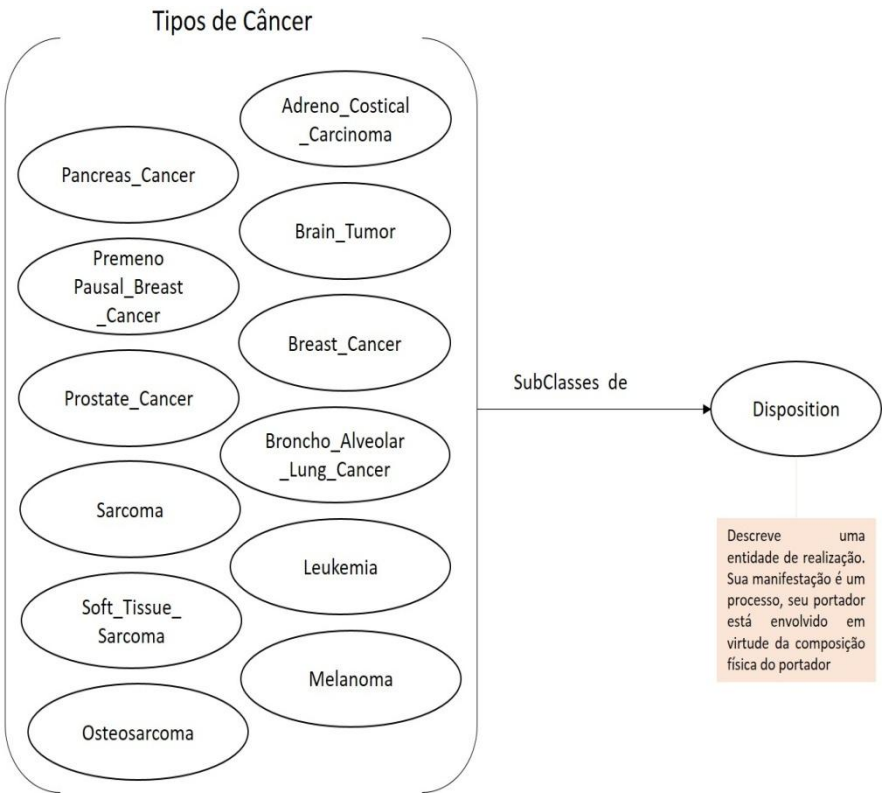


Figura 12: Representação do mapeamento entre DiagsLess60, pertencentes a *Li-Fraumeni Ontology*, com a classe PointInTime, pertencente a BioTop Lite.

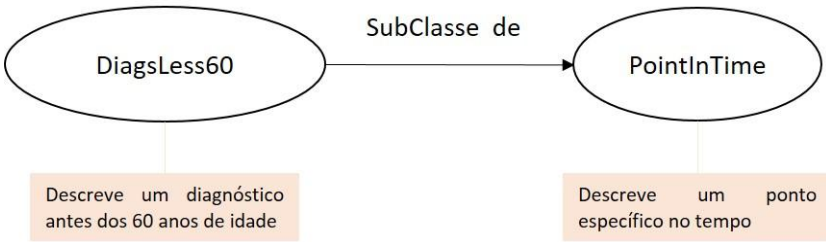
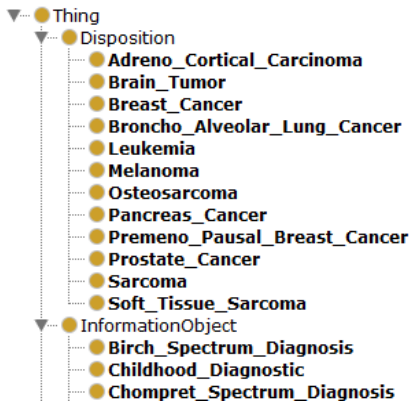


Figura 13: Mapeamento das classes da LFOnto



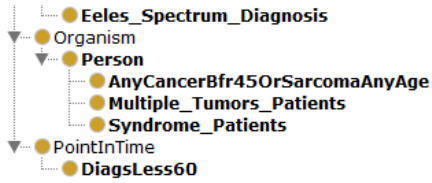


Figura 14: Representação do mapeamento entre a propriedade hasDiagnoticCode, pertencentes a *Li-Fraumeni Ontology*, como equivalente a propriedade 'quality location of' e como subpropriedade da propriedade 'abstract related to', pertencente a BioTop Lite.

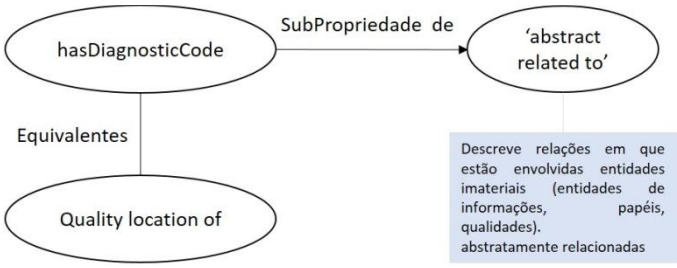


Figura 15: Representação do mapeamento entre a propriedade hasDocument, pertencentes a *Li-Fraumeni Ontology*, como subpropriedade da propriedade 'has patient', pertencente a BioTop Lite.

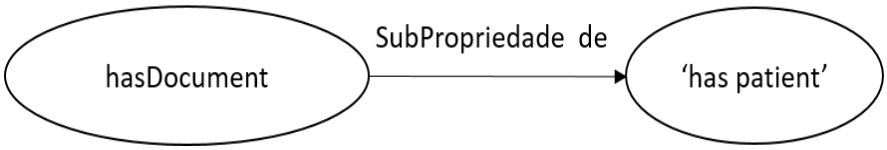
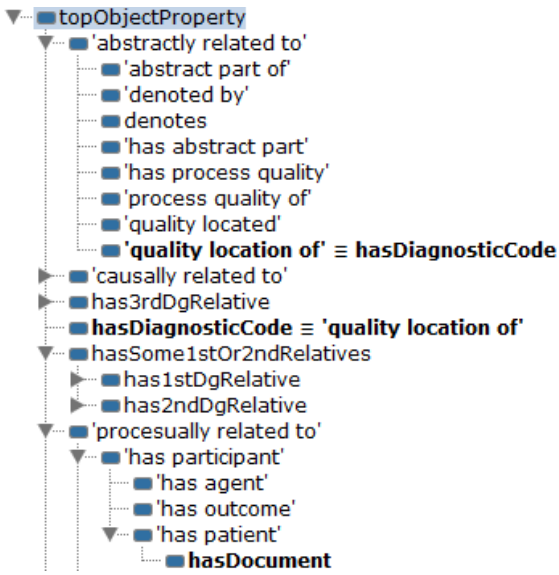


Figura 16: Mapeamento de propriedades da LFOnto



9 Conclusão

Ontologias são especificações formais e explicitas de uma conceitualização compartilhada, visando organizar informações que se tem sobre algum domínio. Diante deste conceito, A *Li-Fraumeni Ontology* visa diagnosticar pacientes que se enquadram nos critérios descritos atualmente para a síndrome de Li-Fraumeni e para ter sua validade comprovada é necessário o mapeamento com uma ontologia de nível superior, que modela o senso comum a ser utilizada por outras ontologias, já utilizada anteriormente e com funcionamento adequado para o domínio em questão. A ontologia escolhida para este processo foi a BioTop Lite, que vem recebendo melhorias, sendo uma camada de alto nível com desempenho consideravelmente melhor que as demais.

O objetivo principal de mapear a *Li-Fraumeni Ontology* à BioTop Lite é validar os termos da primeira ontologia, fazendo o estudo dos vocabulários das três ontologias que a compõe, de acordo com os termos médicos abordados na ontologia de alto nível os quais estes pudessem ser cruzados. O processo de mapeamento consiste em fazer associações entre elementos semelhantes, em cada ontologia, superando desvantagens, fornecendo uma compatibilidade de alto nível e verificação de plausibilidade. O mapeamento das ontologias com a BioTop Lite ocorreu separadamente de forma a manter o modelo de modularização pensado para o seu desenvolvimento. A comprovação do funcionamento deste é feita por inferência do cruzamento das ontologias, onde nenhum deles apresentou qualquer problema ou inconsistências.

Houve uma dificuldade relevante no processo de associação dos termos, no sentido de interpretação das definições dos objetos, pois mesmo que as definições das classes e regras da *Li-Fraumeni Ontology* estejam bem definidas, algumas das classes da BioTop Lite não tem uma definição de fácil entendimento por serem muito genéricas em seu contexto. O que causou muitas vezes dúvidas sobre qual seria a classe mais adequada para o cruzamento e a abrangência das mesmas. Alguns elementos das ontologias estudas não foram mapeados por se achar que não seria adequado defini-los como subclasses de classes da BioTop Lite, outras foram definidas como subclasse para uma classe mesmo deixando dúvidas se ela poderia ser subclasse dessa ou de outra, pois havia, de acordo com a interpretação das definições, a possibilidade de uma mesma classe enquadrar-se como subclasse de duas classes adjuntas, o que não é possível, fazendo com que seja necessário definir apenas uma delas para o cruzamento.

Com relação a outros trabalhos envolvendo mapeamento o presente trabalho apresenta maior clareza quanto ao seu objetivo e a importância que representa. A fundamentação contém tudo o que foi necessário para o trabalho, de conceitos e suas discussões sobre ontologias, onde são usadas e o porquê de serem usadas até o mapeamento, em uma linha de pensamento que permite com que pessoas, que nunca tenham lidado com uma ontologia, consigam chegar até o final deste trabalho entendendo sobre o que leram e principalmente entendendo a sua relevância. Neste trabalho não mapeamos todos os termos presentes na ontologia de Li-Fraumeni, assim como também não validamos o processo com especialistas de domínio. Tendo em vista as dificuldades encontradas durante a pesquisa a próxima etapa será um melhor estudo dos termos da BioTop Lite a fim de buscar associações e ligações, além das que foram feitas, que sejam mais firmes e apresentar a um especialista de domínio. Dessa forma a ontologia de Li-Fraumeni poderá ser ainda mais eficaz no diagnóstico dos pacientes portadores da síndrome.

Referências

[1] GRUBER, T. (1996). “What is an ontology?” [S. l. : s. n.], 1996. Disponível em: < http://www-ksl.stanford.edu/kst/what-is-an-ontology.html >. Acesso em: 17 out. 2015.

[2] GUARINO, N. “Formal ontology, conceptual analysis and knowledge representation”**.** International Journal of Human and Computer Studies, Ed 43, p. 625–640, 1995.

[3] SCHULZ, S., BOEKER, M. “BioTopLite: An *Upper Level Ontology* for the Life Sciences. Evolution, Design and Application”. Informatik 2013. U. Furbach, S. Staab; 2013.

[4] BORST, W. “Construction of Engineering Ontologies for Knowledge Sharing and Reuse”. PhD thesis, University of Twente, P.O. Box 217 - 7500 AE Enschede - The Netherlands, 1997.

[5] ALMEIDA. Mauricio B., BAX. Marcello P. “Uma Visão Geral Sobre Ontologias**:** pesquisa sobre definições, tipos, aplicações, métodos de avaliação e de construção”, artigo. Brasília, 2003.

[6] NOY, N. F; MCGUINNESS, D. L. “Ontology development 101: A guide to creating your first ontology”. <http://protege.stanford.edu/publications/ontology_development/ontology101.pdf> Acessado em: 5 maio. 2016.

[7] GUARINO, N. “Understanding, building and using ontologies”. International Journal of Human and Computer Studies, 45(2/3), 2 1997..

[8] GUIZZARDI, G. “Desenvolvimento para e com reuso: Um estudo de caso no domínio de vídeo sob demanda”. Master’s thesis, Universidade Federal do Espírito Santo, 2000.

[9] MORAIS, E; AMBRÓSIO, A. “Ontologias: conceitos, usos, tipos, metodologias, ferramentas e linguagens”. Goiás, 2007.

[10] HOEHNDORF. Robert. “What is an *Upper Level Ontology*?” http://ontogenesis. knowledgeblog.org/740, 2010.

[11] ACHATZ, Maria Isabel Waddington. “Diagnóstico Molecular da Síndrome de Li-Fraumeni em Famílias Brasileiras”. Dissertação(mestrado) – Fundação Antonio Prudente. São Paulo, 2006.

[12] BUDARUICHE, Ricardo M. S. et al. “*Li-Fraumeni Ontology*: A case study of an ontology for Knowledge Discovery in a Cancer Domain.”. Conference on Intelligent Systems. Brazilian, 2015.

[13] NILES, Ian. “Mapping WordNet to the SUMO Ontology”.<http://www.post-information-age.cz/id32402/jazyk/jazykove(2da/aplikovana(1_lingvistika/Ontologie/WordNet/nilesWordNet.pdf> Acessado em: 28 junho. 2016.

[14] REED, Stephen L., LENAT, Douglas B. “Mapping Ontologies into Cyc”. <https://www.aaai.org/Papers/Workshops/2002/WS-02-11/WS02-11-010.pdf > Acessado em: 28 junho. 2016.

[15] MIYOSH, Newton Shydeo Brandão, FELIPE, Joaquim Cezar. “Ontologia para o Registro de Evolução Clínica do Paciente”. <https://uspdigital.usp.br/siicusp/cdOnlineTrabalhoVisualizarResumo?numeroInscricaoTrabalho=3218&numeroEdicao=17> Acessado em: 28 junho. 2016.

Captura de tráfego de rede e classificação de fluxos utilizando aprendizado de máquina

Patrícia Dayana de Araújo Souza[1]
Kerllon Fontenele de Andrade [2]
José Vigno Moura Sousa[3]

Resumo: O aumento do número de usuários aliado à qualidade de transmissão e velocidade de links de internet implica diretamente no volume de dados trafegados em uma rede, a captura e análise desses dados podem trazer informações importantes para a prevenção de problemas. Uma classificação de fluxos de tráfego é uma tarefa determinante, podendo ser aplicada para diversos objetivos, tais como, criar mecanismos para detecção de fluxos de tráfego maliciosos, corte de tráfego que consome maior quantidade de banda ou prejudique o bom funcionamento da rede. Nesse sentido, esse trabalho tem como objetivo obter o tráfego gerado da rede da Universidade Estadual do Piauí, campus Piripiri, a fim de criar um modelo classificador de tráfego utilizando aprendizado de máquina. Para tanto, utilizou-se a ferramenta wireshark na intenção de capturar os pacotes de rede e a plataforma de mineração de dados Weka com o proposito de classificar os perfis de tráfego fazendo uso dos algoritmos de aprendizagem implementados na plataforma.

Palavras-chave: Aprendizagem de máquina. Captura. Tráfego de rede.

Abstract: *The increase in the number of users combined with the quality of transmission and Internet links speed directly implies the volume of data traffic on a network, the capture and analysis of such data can provide important information to prevent problems. A classification of traffic flows is a crucial task and can be applied for various purposes, such as creating mechanisms to detect malicious traffic flows, cutting traffic that consumes greater amount of bandwidth or detrimental to the proper functioning of the network. Thus, this study aims to get traffic generated network of the State University of Piauí Piripiri campus in order to create a classifier model traffic using machine learning. For this, we used the Wireshark tool in an attempt to capture the network packets and data mining platform Weka with the purpose of classifying traffic profiles making use of learning algorithms implemented on the platform.*

Keywords: *Machine learning programs. Catch. Network traffic.*

1 Introdução

As redes de computadores atuais são compostas por uma grande variedade de dispositivos. A internet, sua maior representante, é um imenso número de redes ao redor do planeta, que se comunica através do protocolo TCP/IP. Baseia-se na tecnologia cliente/servidor, onde um cliente utiliza a internet para requisitar informações de um servidor e este servidor envia a informação requerida de volta ao cliente via Internet. Esse tipo de comunicação permite a execução de diversas aplicações.

O tráfego de uma rede consiste em um conjunto de pacotes de rede gerados durante as comunicações entre cliente/servidor e entre máquinas (hosts). Já os Fluxos das aplicações podem ser entendidos como uma sequência de pacotes transmitida entre dois hosts, onde esses pacotes devem possuir características semelhantes, tais como: IP (origem e destino), porta (origem e destino), e tipo protocolo da camada de transporte. Entender o

[1]Bacharelando em Ciência da Computação pela Universidade Estadual do Piauí - UESPI/Piripiri. {p.dayanasouza1293@hotmail.com}
[2]Mestre em Informática Aplicada, professor temporário da Universidade Estadual do Piauí - UESPI/Piripiri. {kerllonandrade@hotmail.com}
[3]Mestre acadêmico em Ciência da Computação, professor efetivo da Universidade Estadual do Piauí - UESPI/Piripiri. {josevigno@gmail.com}

comportamento dos fluxos de comunicação das aplicações pode auxiliar na tomada de decisão de difíceis problemas de gerência de rede, tais como: bloqueio ou limitação de aplicações ou serviço consome mais da banda da internet, prevenção de invasões à rede interna e melhoria na própria gerencia e dos ativos da rede. Assim, com uso de uma ou mais ferramentas que auxiliem no levantamento dessas informações, um administrador de rede ou provedor desse serviço pode garantir o chamado QOS (Qualidade de serviço).

O grande responsável pela comunicação mundial entre os indivíduos é a internet e, consequentemente, as redes de computadores, sejam elas redes cabeadas ou redes sem fio. Com aumento do número de usuários aliado à qualidade de transmissão e velocidade de links de banda larga de internet, o volume de dados trafegados nas redes aumentou progressivamente. De acordo com Barros [1], a internet sofre sérios problemas técnicos, relativos ao estabelecimento e garantia de conexões, e pode enfrentar nos próximos anos uma mudança do seu paradigma atual de comunicações.

Há inegavelmente uma necessidade de se supervisionar e controlar o funcionamento do tráfego gerado em uma rede, para melhorar, por exemplo, a velocidade da rede e o desempenho das máquinas, pois além do fator produtividade, o tráfego de rede precisa ser balanceado entre aplicações, para que os sistemas que rodam nela não fiquem lentos para quem está utilizando.

Saber o comportamento de em uma determinada rede é essencial para identificação do mau uso de aplicações Web e para a prevenção de seu baixo desempenho. Por exemplo, jogos online, youtube, compartilhamento de arquivos via aplicações P2P, redes sociais, entre outros, são responsáveis por um grande consumo da largura de banda de uma rede afetando diretamente no desempenho da rede e na produtividade de empresas, organizações e instituições.

Nesse contexto, coletar dados e, especificamente, extrair informações dos fluxos referentes às aplicações e classifica-las é de fundamental importância, permitindo, assim, o uso do processo KDD (Descoberta de conhecimento), que pode ser utilizado para auxiliar a descoberta de conhecimento útil em bases de dados, além do uso de técnicas de Aprendizado de Máquina como forma de automatizar esse processo. No entanto, com intuito de uma consistência desses dados, faz-se necessário, por vezes, a manipulação e/ou rotulação desses dados. A metodologia proposta desse trabalho distinguiu essas tarefas da seguinte forma: Coletar do tráfego da rede com o uso do Sniffer Wireshark[4], uso da ferramenta Splitcap[5] para separar os fluxos e os subfluxos das aplicações, uso da biblioteca jpcap[6] em conjunto com a linguagem java para o desenvolvimento de uma aplicação para manipular e rotular os arquivos de captura (pcap[7]) e, por fim, classificar os perfis de tráfego com o uso da plataforma de mineração de dados Weka[8].

Os testes foram realizados com a utilização do tráfego oriundo da rede da Universidade Estadual do Piauí, campus Piripiri, em que se obteve um total de 06 (seis) classes de aplicação no pré-processamento dos dados e apresentaram na classificação uma precisão de 94,15% para subfluxos TCP e 99% para subfluxos UDP. Na estratégia apresentada, o número de atributos variou, respectivamente, de 10 a 13 para UDP e TCP.

Este trabalho está organizado da seguinte maneira: a seção 2 apresenta uma breve revisão sobre trabalhos relacionados ao tema deste artigo; uma introdução sobre aprendizado de máquinas e classificadores e suas aplicações em classificação de tráfego é demonstrada na seção 3; na seção 4 é explanado o procedimento para coleta, pré-processamento e rotulação de fluxos utilizados neste trabalho bem como uma análise dos resultados obtidos; a conclusão é apresentada na seção 5 com as principais considerações e os direcionamentos para trabalhos futuros.

2 Trabalhos Relacionados

Para Hu e Shen [2], classificação de tráfego usando técnicas de Aprendizagem de Máquina (ML) tem sido um ativo e difícil tópico de investigação dos últimos anos. Alguns pesquisadores obtêm bons resultados em suas pesquisas.

[4] https://www.wireshark.org/.
[5] https://www.netresec.com/?page=SplitCap
[6] http://netresearch.ics.uci.edu/kfujii/Jpcap/doc/
[7] http://www.reviversoft.com/pt/file-extensions/pcap
[8] http://www.cs.waikato.ac.nz/ml/weka/

Diversos trabalhos têm abordado o tema mineração de dados voltados para mineração de sessão do tráfego, tal como Santos et al.[3], que aborda a caracterização do comportamento padrão do tráfego HTTP, através das técnicas de Análise de Séries Temporais e Inteligência Computacional, utilizando-se uma metodologia para caracterizar o comportamento padrão do tráfego de rede por meio da analise de atributos das sessões do tráfego, aplicando-se, assim, a técnica de mineração de dados por clusterização. Já Santos [4], caracteriza o tráfego padrão de rede TCP/IP gerado a partir de serviços Web, onde cada registro de sessão é descrito por nove atributos para treinamento satisfatório de classificadores a fim de detectar anomalias. Sua metodologia consiste em análise, processamento e extração da informação de tráfego de redes de computadores, utilizando-se técnicas de Inteligência Computacional, em uma abordagem de mineração de dados, com o intuito de identificar padrões de anomalias nos dados espaço-temporais do fluxo de rede observado.

O tema de classificação do tráfego de rede sob uma perspectiva de classificação de fluxos e de seleção de variáveis para uma classificação também tem chamado à atenção de diversos pesquisadores, tal como Vilela [5], que defende a importância da técnica de caracterizar o tráfego medindo os fluxos de comunicação, de tal modo, sua metodologia propõe a análise de três variáveis: tamanho, duração e taxa; além de separar os fluxos em classes (grandes, pequenos e intermediários), desse modo, traçou-se um perfil de fluxos para melhor conhecer as características do tráfego de rede capturado.

Moore e Zuev [6], aplicam um algoritmo de aprendizado de máquina supervisionado para classificar o tráfego de rede por aplicação, neste caso foram aproveitados os dados de fluxos, demonstrando na classificação resultante de tráfego, um bom desempenho tanto em termos de precisão e quanto confiança.

Nos estudos propostos por Li et al. [7], o algoritmo Naive Bayes é utilizado para a classificação de tráfego IP baseada em propriedades estatísticas dos fluxos, tais como o tamanho médio do segmento, a variância do tamanho de carga e tamanho da janela inicial; um total de 10 variáveis foi selecionado e uma taxa de acerto de 96% foi obtida na classificação do tráfego de 10 classes de aplicação (avaliados apenas fluxos TCP).

Na pesquisa de Ribeiro [8] é descrito uma estratégia de classificação baseada em subfluxos aplicada aos tráfegos TCP e UDP, a proposta utiliza a decomposição de 1CT sobre variáveis estatísticas dos subfluxos, as quais são obtidas somente a partir da informação dos cabeçalhos dos pacotes.

Andrade [9], defende uma abordagem de classificação de fluxos P2P baseada em aprendizado de máquina que utiliza abordagens heurísticas combinadas a valores estatísticos da camada de transporte para a extração de atributos, onde os atributos utilizados na abordagem são escolhidos e aplicados ao algoritmo de Árvore de Decisão e ao algoritmo Naive Bayes. Esse trabalho apresentou, ainda, uma nova estratégia para rotulação de pacotes baseadas em ferramentas livres: SplitCap, tcpdump, Wireshark e a biblioteca Jpcap.

Neste trabalho fará uso de algumas estratégias propostas nos trabalhos anteriores, em especial, o que fora proposto por Moore e Zuev [6], Li et al. [7] e Andrade [9]. Terá como base de dados para o pré-processamento, o tráfego colhido da rede da Universidade Estadual do Piauí, Campus Piripiri, usando-a de uma forma não intrusivo na obtenção dos dados.

3 Classificação usando aprendizado de máquina

As técnicas de mineração de dados podem ser aplicadas a diversas tarefas, tais como: classificação, estimativa, associação, entre outras. Contudo, toda técnica de mineração passa por um processo chamado de treinamento, essa fase de treinamento tem este nome por ser uma técnica de apresentação dos dados processados para o algoritmo de mineração, cujo objetivo é identificar, ou seja, “aprender” as características ou padrões úteis ao objetivo do processo de descoberta de conhecimento.

Segundo Mitchell [10], um programa de computador aprende a partir da experiência **E** com relação a alguma classe de tarefas **T** e medida de desempenho **D**. Portanto, pode-se dizer que um programa de computador é capaz de “aprender” se seu desempenho em tarefas de **T**, medidas por **D**, melhora com a experiência **E**. Para que se tenha um problema de aprendizado bem definido é necessário identificar três características: a classe de tarefas, a medida de desempenho a ser melhorada e a fonte de experiências. Distinguem-se estas três características na classificação proposta da seguinte forma: Tarefa T: identificar (classificar) tráfego TCP e UDP; Medida de desempenho D: porcentagem de pacotes corretamente classificados; e experiência de treinamento E: um banco de dados com tráfegos TCP e UDP corretamente rotulados.

Em um processo de decisão, seja ele qual for, é necessário, ainda, tomar algumas medidas que orientem o caminho a seguir, ou quanto à decisão a ser tomada. Existem diversas medidas que podem ser consideradas no aprendizado de máquina, tais como: acurácia/erro (taxa de predições corretas ou incorretas realizada pelo modelo para um determinado conjunto de dados), taxa de falsos positivos e/ou de verdadeiro positivos (onde dado um classificador para discriminar classes X e Y, supondo que X é a classe mais relevante ou positiva, o número de falsos positivos é a quantidade de instâncias da classe Y classificadas como da classe X; do mesmo modo, o número de falsos negativos é a quantidade de exemplos da classe X classificados como da classe Y), precisão (número de verdadeiros positivos ou número de itens corretamente classificados como pertencentes à classe positiva) etc.

Tabela 1: Técnicas e algoritmos implementadas no Weka. (Fonte: Lima [11]).

Técnicas de classificação	**Algoritmo(s)**
Bayes (probalístico)	*NaiveBayes, NaiveBayesSimple*
Function (técnica de regressão linear e logística	*LinearRegression, logistic, SMO, VotedPerceptron*
Lazy (técnica baseada em instâncias)	*IN1, IBk, KStar*
Meta (regressão por discretização)	*AdaBoostM1, AdditiveRegression, AttributeSelectedClassifier, Bagging, ClassificationViaRegression, CostSentiveClassifer, CVParameterSelection, etc.*
Misc (técnica de discretização)	*Hiperpipes, VFI*
Rules (regras de decisão)	*J48.PART, DecisionTable, OneR.*
Trees (árvores de decisão)	*ADTree, DecisionStump, Randon Forest, Id3, J48.*

A tabela 1 representa alguns das técnicas e os respectivos algoritmos de Aprendizado de Máquina que o Weka disponibiliza para classificação de instancias. Serão comentados nas próximas seções, com ênfase, os algoritmos de classificação por Regras e Árvores de Decisão, uma vez que apresentaram melhores resultados na predição do conjunto de dados obtidos nesse trabalho.

3.1 Classificador *Decision Table*

O algoritmo Decision Table baseia-se em regras que representam o conhecimento na forma de tabela. De acordo com De Souza [12], Decision Table apresenta como resultado da classificação os dados processados em forma de tabelas de decisão, a qual é uma matriz que representa as condições e ações combinadas, onde as condições são um conjunto de informações utilizadas para tomar as decisões.

"Algumas tabelas utilizam valores verdadeiro ou falso (true ou false) para representar as alternativas condições (estilo if-then-else). Outras utilizam de alternativas numeradas (estilo switch-case) e, ainda, existem aquelas que se utilizam da lógica fuzzy (lógica difusa que apresenta vários níveis entre verdadeiro e falso) ou de representações probabilísticas para as alternativas condicionais" Pellucci et al. [13].

Nesse sentido, uma decisão pode ser entendida como uma ação que só poderá ser realizada caso a condição seja satisfeita, onde as ações são as operações que deverão ser executadas após a realização das combinações de condições. A partir do resultado das ações efetuadas através das condições são criadas as regras, que podem ser denominadas de Regras de Decisão.

3.2 Classificador *Trees Randon Forest*

Segundo Onoda [14], as árvores de decisão são modelos estatísticos utilizados em aprendizados supervisionados, nas quais um conjunto de atributos é utilizado para predizer a classificação de uma instância. As árvores de decisão representam um exemplo de aprendizado indutivo, e esse tipo de algoritmo cria uma hipótese baseada em instâncias particulares para, a partir dessa hipótese, criar conclusões gerais.

Para classificar uma instância pertencente a uma determinada classe, um algoritmo de árvore de decisão constrói uma árvore e a percorre a partir da raiz até o nó folha, que representa a classificação desta instância. O processo de decisão de qual será o próximo nó a ser analisado é semelhante à estrutura if-then, em que o algoritmo usa uma condição para decidir qual o próximo nó a ser analisado. Esse processo se repete até que o algoritmo chegue a uma folha e, assim, tenha uma classificação.

De acordo Breiman [15], uma floresta aleatória (random forest) é uma combinação de árvores de decisão, em que cada árvore depende dos valores de vetores aleatórios amostrados de forma independente e distribuídos igualmente para todas as arvores na floresta. Nesse método, depois que um determinado número de arvores são geradas, cada uma lança um voto para uma classe do problema, considerando um vetor de entrada. Então, a classe mais votada será escolhida na predição do classificador.

4 Coleta, pré-processamento e rotulação de fluxos

O tráfego de uma rede consiste em um amplo conjunto de dados de pacotes de rede gerados durante as comunicações entre os hosts. Uma técnica que vem sendo frequentemente utilizada por administradores de rede é a "captura de pacotes" através de aplicações que realizam essa tarefa, visto que possibilita a monitoração de redes. McCanne e Jacobson [16], apresentam uma arquitetura para a captura de pacotes, conhecida como "BPF - BSD Packet Filtering", que realiza a coleta e o armazenamento de vários pacotes. Além disso, o BPF encapsula os dados capturados para cada pacote com um cabeçalho próprio, que contém o horário de captura, tamanho e deslocamento para o alinhamento dos dados. A biblioteca libpcap implementa a atual arquitetura BPF e a continuidade de seu desenvolvimento é de responsabilidade do grupo tcpdump.org (que detém o sniffer tcpdump, baseada em linha de comando).

A libpcap[9] é utilizada por uma infinidade de outras ferramentas, como por exemplo, o wireshark, que, de acordo com os desenvolvedores da ferramenta, "é o analisador de protocolos de rede mais popular do mundo". Trata-se de uma ferramenta utilizada por profissionais das áreas de rede e segurança da informação e por desenvolvedores, disponível gratuitamente com código fonte aberto, possuindo função de analisador de protocolos de rede e manipulação de arquivos no formato pcap capturados a partir de qualquer sniffer que gere arquivos de captura nesse formato, considerada por Galvão [17] como sua principal função.

O processo de coleta do tráfego de rede consiste na captura contínua de pacotes de rede contendo os dados brutos (em formato hexadecimal). De acordo com Vilela [5], uma importante técnica para caracterização do tráfego é a medição dos fluxos de comunicação, onde fluxo pode ser definido como uma sequência de pacotes transmitida entre dois hosts com as mesmas características. Com base nas informações dos cabeçalhos dos pacotes é possível agrupá-los em fluxos comuns, o que permite responder vários aspectos do tráfego: quem, o que, quando, onde e como.

A coleta, pré-processamento e rotulação de fluxos está aqui descrito em fases e nas etapas constituintes dessas fases para melhor entendimento da proposta, porém em resumo, a figura 1 esboça como esse processo fora feito.

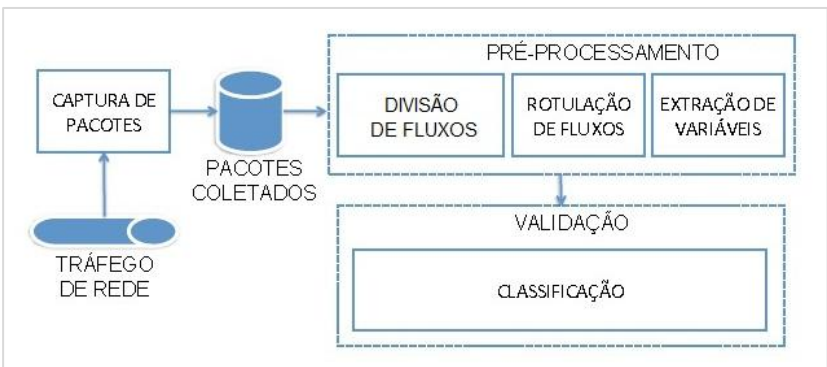


Figura 1: Resumo das fases da construção da proposta.

[9] https://wiki.wireshark.org/libpcap

4.1 Primeira fase: obtenção e pré-processamento dos dados

Para a obtenção dos dados para análise, fez-se necessário a configuração Port mirroring (espelhamento de portas) em um dos switchs que distribui a rede em estudo, para que assim, o Sniffer utilizado (wireshark) pudesse capturar todo o tráfego que passa pelo switch.

O pré-processamento dos dados inicia-se à medida que os dados são coletados e organizados na forma de um conjunto de dados. Pôs-se, então, a ferramenta em modo promíscuo para capturar todo o tráfego da rede durante 02 (dois) meses, em dias alternados e em horários de pico. Após a coleta dos pacotes (pcap) gerados pelo sniffer, o pré-processamento organizou-se nas seguintes etapas:

• 1º Etapa (divisão dos pacotes coletados (pcap) em fluxos): utilizou-se a ferramenta SplitCap, cujo obteve-se todos os fluxos divididos em arquivos individuais. Os fluxos são separados em fluxos TCP e fluxos UDP. A figura 2 exemplifica como se deu essa etapa.

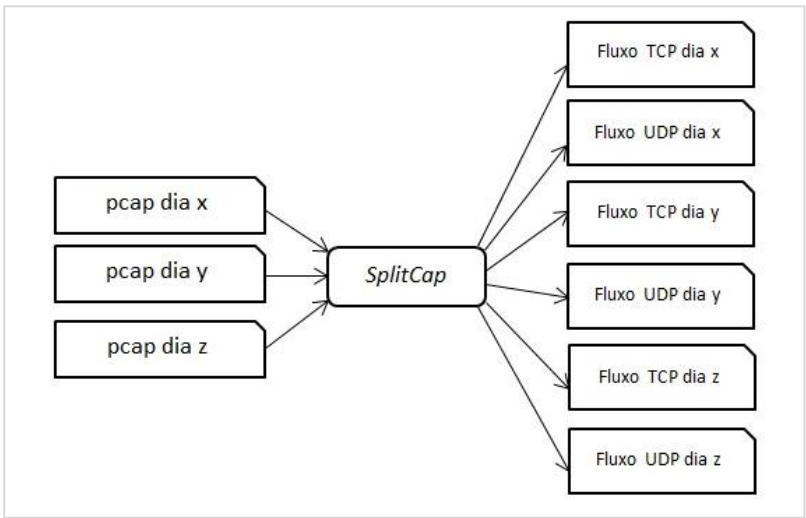


Figura 2: Divisão dos pacotes (pcap) em fluxos TCP e fluxos UDP.

• 2º Etapa (decomposição dos pacotes coletados (pcap) por protocolo): o wireshark dispõe da opção Filter, a qual é possível "filtrar" a captura e/ou análise por portas ou protocolos, em que se conseguiu fluxos contendo as assinaturas (sequência de Bytes/strings e offset que são únicos de cada aplicação). Abaixo, a figura 3 demonstra os passos dessa etapa.

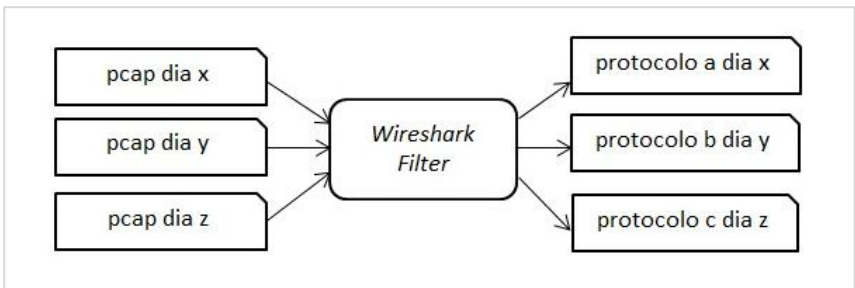


Figura 3: Decomposição dos pcap por protocolo.

• 3º Etapa (divisão em fluxos de cada protocolo, chamados aqui de subfluxos): utilizou-se a ferramenta SplitCap novamente. Em seguida, a figura 4 ilustra tal etapa.

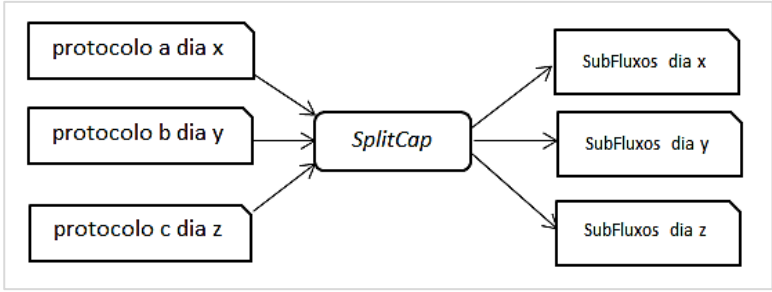


Figura 4: Divisão em SubFluxos de cada protocolo.

4.2 Segunda fase: desenvolvimento de uma aplicação para comparar os fluxos e rotular em classes

No intuito de uma consistência dos dados para mineração e posterior classificação, fez-se necessário o desenvolvimento de uma aplicação em java, utilizando a biblioteca jpcap, que permite a leitura e manipulação de arquivos pcap. Tal aplicação tem como objetivo, a princípio, comparar os fluxos obtidos na primeira etapa da fase anterior do projeto com os subfluxos da terceira etapa dessa mesma fase, os quais dizem respeito aos próprios arquivos de capturas da primeira fase, porém, com as assinaturas de cada protocolo.

O resultado dessa comparação consiste na saída de fluxos corretamente rotulados por cada protocolo especificado. A razão para tal é porque o filtro captura apenas os pacotes que contêm a assinatura de um protocolo, e por isso é preciso comparar os fluxos com os subfluxos para que se alcancem os fluxos completos. Então, comparando o IP Origem dos fluxos com IP Origem dos Subfluxos e Porta Origem dos fluxos com Porta Origem dos Subfluxos, bem como IP e Porta destino de ambos, resulta nos fluxos completos.

Para entender o que é feito no segundo momento dessa aplicação, é importante absorver o ponto de vista de Ribeiro [8], que defende: “abordagens baseadas em aprendizado de máquina partem da premissa de que amostras de uma mesma classe possuem um conjunto de variáveis com valores próximos, onde uma variável pode ser qualquer atributo relevante para a predição de um conjunto de classes”. Como já se obteve os fluxos rotulados, isso pode ser entendido como as classes citadas por Ribeiro [8], prontas para serem retirados os atributos. Segue abaixo (figura 6), um fluxograma da lógica seguida na aplicação:

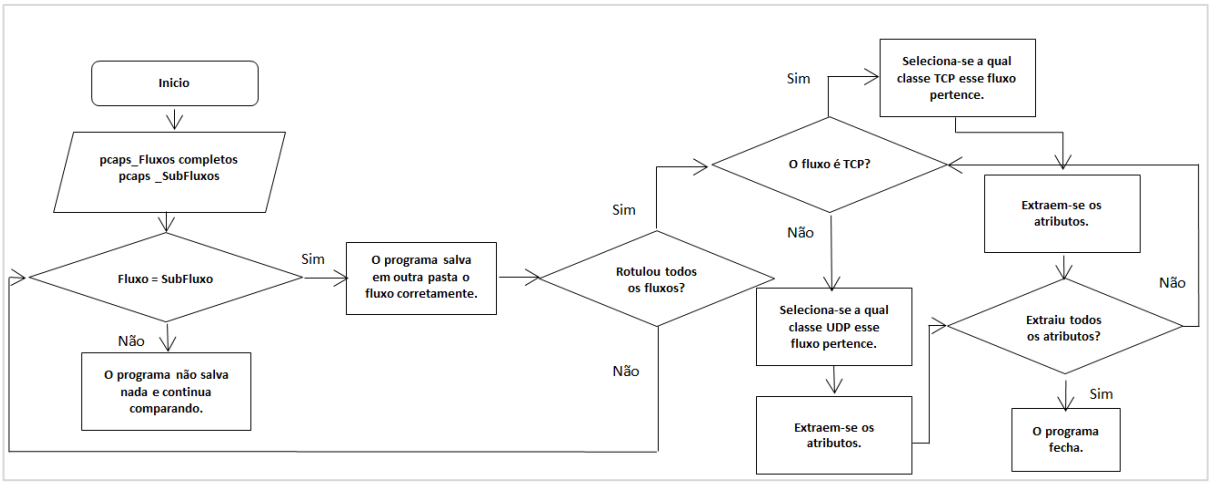


Figura 5: Fluxograma da lógica seguida na aplicação.

Na literatura é possível encontrar vários trabalhos que tratam da seleção das variáveis, dos quais são obtidas a partir da informação dos cabeçalhos dos pacotes, sem explorar a payload para classificação utilizando aprendizado de máquina. A aplicação desenvolvida foi capaz de estipular 17 atributos estatísticos para a predição das classes TCP (comprimento do pacote, média do comprimento do pacote, quantidade de bytes do cabeçalho, byte de maior valor, byte de menor valor, tamanho janela, media janela, tamanho da payload, média do tamanho

da payload, comprimento do cabeçalho, média comprimento, número de sequencia, média do número de sequencia, número de ACK, media do número de ACK, porta de origem e porta de destino).

Já para as classes UDP foram estipulados 10 atributos (comprimento do pacote, quantidade de bytes do cabeçalho, byte de maior valor, byte de menor valor, tamanho da payload, média do tamanho da payload, comprimento do cabeçalho, média comprimento, porta de origem e porta de destino). Esse conjunto de atributos extraído dos pacotes constitui-se o Dataset, que foi utilizado em conjunto com a plataforma Weka para a classificação dos perfis de tráfego obtidos das classes TCP e UDP.

4.3 Terceira fase: discussão dos resultados obtidos na classificação

Utilizou-se a técnica Cross-validation (validação cruzada) na classificação da plataforma Weka, na qual se testou (tanto pra TCP como para UDP) os classificadores Trees (Árvores de decisão), Rules (Regras de decisão), Lazy learning (aprendizado preguiçoso), Bayes (probabilístico) e Function Multilayer Perceptron (Rede Neural). Abaixo, destacam-se as métricas utilizadas para avaliar o desempenho do método proposto:

- Verdadeiro Positivo: quantidade de fluxos corretamente classificados;
- Falsos Positivos: quantidade de fluxos erroneamente classificados;
- Precisão: representa o percentual total dos fluxos corretamente classificados dentre todos os fluxos.

O weka possibilita, dentre outras funcionalidades, remover atributos que não têm grande relevância ou que são redundantes na classificação (select attributes). Dessa forma, à medida que testado os classificadores, houve a necessidade da remoção de alguns atributos para que a acurácia (taxa de precisão) obtivesse melhores resultados. Nesse seguimento, diminuíram-se de 17 para 10 atributos correspondente às classes TCP, o que permite uma melhora na taxa de precisão, principalmente quando utilizado o classificador Trees Randon Forest com acurácia de aproximadamente de 94%, como pode ser visto no gráfico 1. Já para as classes UDP, não houve a retirada de atributos, pois ao ser aplicado regras de decisão (Rules Decision Table), alcançou-se uma taxa de acerto de aproximadamente 99.15%, como demostrado no gráfico 2.

Gráfico 1: Precisão dos classificadores com relação à quantidade de Atributos (Classes TCP).

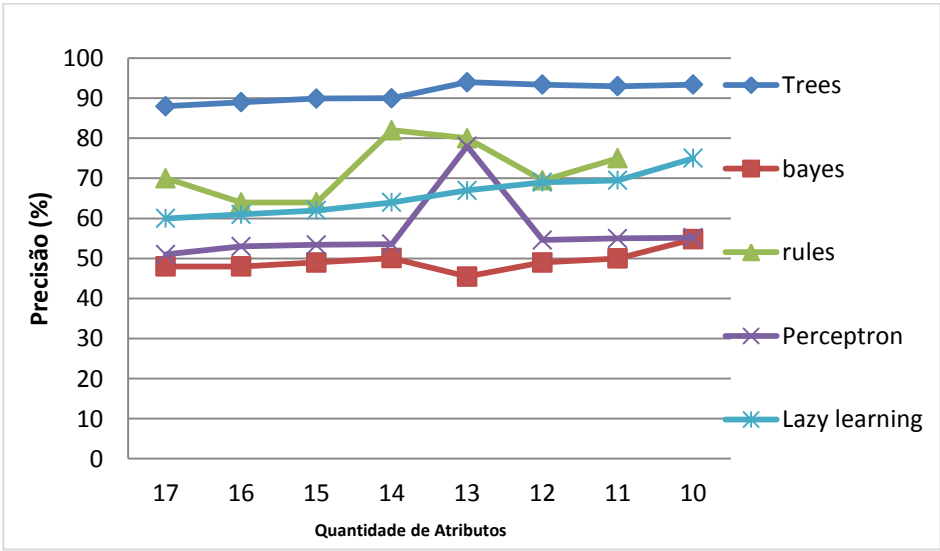
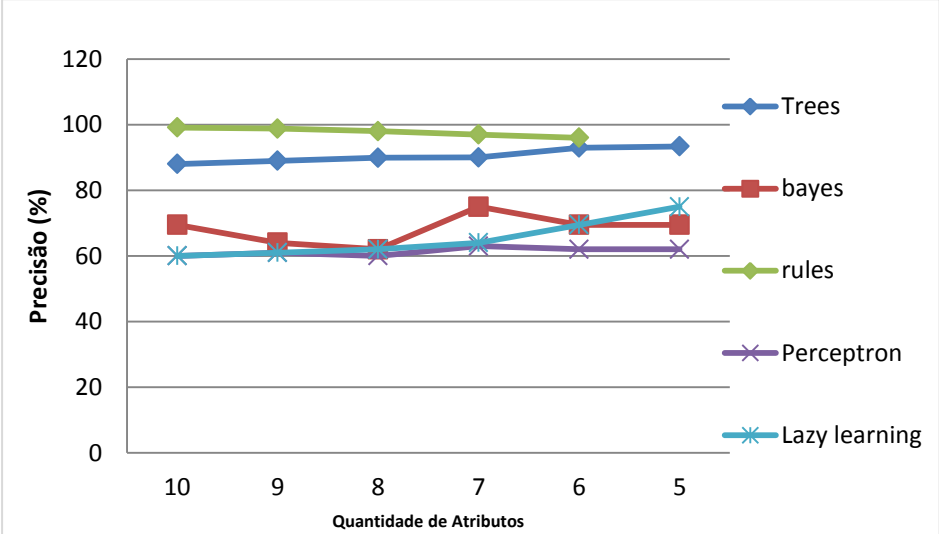


Gráfico 2: Precisão dos classificadores com relação à quantidade de Atributos (Classes UDP).



As tabelas 2 e 3 representam, respectivamente, as matrizes de confusão das Classes TCP e UDP que o Weka apresentou logo após a classificação das instancias com os melhores classificadores (Randon Forest e Decision Table). A matriz de confusão oferece uma medida efetiva do modelo de classificação, ao mostrar o número de classificações corretas versus as classificações preditas para cada classe, sobre um conjunto de instâncias. O número de acertos, para cada classe, se localiza na diagonal principal M(Ci,Ci) da matriz e o demais elementos M(Ci,Cj), para $i \neq j$, representam erros na classificação. A matriz de confusão de um classificador ideal possui todos esses elementos iguais à zero uma vez que ele não comete erros.

Tabela 2: Matriz de confusão das classes TCP do melhor classificador (Randon Forest).

Classes	**p2p**	**ftp**	**http**	**ssl**
p2p	591	2	23	18
ftp	6	888	0	1
http	10	9	1009	72
ssl	10	1	71	1018

Tabela 3: Matriz de confusão das classes UDP do melhor classificador (Decision Table).

Classes	**dns**	**quick**
dns	1009	7
quick	11	1089

Dessa forma obteve-se resultados superiores quando comparados aos trabalhos da literatura, entre os quais se menciona Filho et al. [18], que em sua abordagem, o algoritmo um-contra-todos (1ct) é aplicado em duas estratégias de classificação online baseadas em variáveis estatísticas para subfluxos TCP e UDP, contendo um total de 07 (sete) classes de aplicação. Os resultados obtidos apresentaram uma precisão de 98,43% para

subfluxos iniciais TCP e 98,75% para subfluxos UDP. Resultados, consideravelmente, relevantes e bem próximos dos resultados obtidos na metodologia proposta nesse trabalho, que por sua vez adquiriu uma taxa de acerto de aproximadamente de 94% quando aplicado o algoritmo Trees Randon Forest para as classes TCP e uma precisão de aproximadamente 99.15% para as classes UDP ao ser aplicado Rules Decision Table, um resultado relativamente superior.

Em Ribeiro et al. [19], igualmente a Filho et al. [18] também aplicou a abordagem 1ct para classificação de tráfego TCP e UDP. A metodologia apresentada utiliza N classificadores binários para um problema com N classes de tráfego. Um processo de seleção de variáveis e tamanhos ótimos de subfluxos é realizado para os subfluxos iniciais e aleatórios de cada classe, onde foram adquiridas taxas de acerto de 98.45% e 94.24%, respectivamente para subfluxos iniciais e aleatórios. Um total de 05 (cinco) classes de tráfego TCP foram analisadas neste trabalho. Já para as classes de tráfego UDP, uma taxa de acerto equivalente quando aplicado o algoritmo Decision Table da proposta deste trabalho, pois aproximadamente 99% foi obtida para os subfluxos UDP.

No trabalho proposto em Li et al. [7], o algoritmo Naive Bayes é utilizado para a classificação de tráfego IP baseada em propriedades estatísticas dos fluxos. Um total de 10 variáveis foi selecionado e uma taxa de acerto de 96% foi obtida na classificação do tráfego de 10 (dez) classes de aplicações TCP, no entanto, tal trabalho, foca apenas os fluxos TCP.

5 Considerações finais

Neste trabalho discutiu-se a importância em ter conhecimento sobre o tráfego que flui em uma rede, pois tal conhecimento proporciona entendimento no que trafega em uma rede e quais medidas ou precações podem ser tomadas na medida em que surgem problemas. Propõe-se uma abordagem de classificação de fluxos TCP e UDP utilizando aprendizado de máquina com uso de atributos estatísticos referentes aos fluxos para ajudar na tomada de decisões difíceis. Na metodologia proposta fora adquirido uma taxa de certo de aproximadamente 94% quando aplicado o algoritmo Trees Randon Forest para as classes TCP, e uma precisão de aprox. 99.15% para as classes UDP ao ser aplicado Rules Decision Table

A continuidade deste trabalho segue em três direções: investigar o desempenho da abordagem proposta para novas classes de aplicações e investigar o desempenho de outros classificadores com novas variáveis para a classificação de tráfego.

Referências

[1] BARROS, MICHAEL T. A. O. Classificação de fluxos IP como ferramenta para engenharia de tráfego na internet. Dissertação (Pós-Graduação em Ciência da Computação) - UFCG, Campina Grande, PB. 2012.

[2] HU, BIN and SHEN, YI. Machine learning based network traffic classification: a survey. Journal of Information and Computational science 9.11. 2012.

[3] SANTOS, ADRIANA C. F. SILVA, JOSE D.S. SÁ SILVA, LÍLIA. SENE, MILENA P.C. Análise de séries temporais para caracterizar o tráfego de rede utilizando mineração de dados por clusterização. X Workshop dos Cursos de Computação Aplicada, INPE, São José dos Campos, SP. 2010.

[4] SANTOS, ADRIANA C. F. Uma metodologia para caracterização do tráfego de redes de computadores: uma aplicação em detecção de anomalias. Tese (Doutorado em Computação Aplicada) - INPE, São José dos Campos, SP. 2011.

[5] VILELA, GUILHERME S. Caracterização de tráfego utilizando Classificação de fluxos de comunicação. Dissertação (Mestrado em Ciências em Engenharia de sistemas e computação) – UFRJ, Rio de Janeiro. 2006.

[6] MOORE, ANDREW W. and ZUEV, DENIS. Internet Traffic Classification Using Bayesian Analysis Techniques. Technical Report, Queen Mary University of London, 2005.

[7] LI, WEI. ABDIN, KAYSAR. DANN, ROBERT and MOORE, ANDREW W. Approaching Real-time Network Traffic Classification. Technical Report, Queen Mary University of London. 2006.

[8] RIBEIRO, V. P. Classificação de Tráfego Online Baseada em Sub-fluxos. Dissertação (Mestrado em Informática Aplicada) - UNIFOR, Fortaleza - CE. 2011.

[9] ANDRADE, KÉRLLON F. Discriminação de tráfego P2P utilizando Árvores de Decisão e Naive Bayes. Dissertação (Mestrado em Informática Aplicada) — UNIFOR, Fortaleza – CE. 2014.

[10] MITCHELL, TOM M. Machine Learning. McGraw-Hill, USA. 1997.

[11] LIMA, TÂNIA S. Estudo comparativo dos algoritmos de classificação da ferramenta Weka. Trabalho de conclusão de curso (Estágio supervisionado em Sistemas de Informação) – ULBRA, Palmas - TO. 2005.

[12] DE SOUZA, OTÁVIO R. M. Mineração de Dados de um Plano de Saúde para Obter Regras de Associação. Dissertação (Mestrado em Engenharia de Produção) – UFSC, Santa Catarina – SC. 2000.

[13] PELLUCCI, PAULO R. S. DE PAULA, RENATO R. SILVA, WALTER B. O. LADEIRA, ANA P. Utilização de técnicas de aprendizado de máquina no reconhecimento de entidades nomeadas no português. Revista e-xacta (ISSN: 1984-3151). 2011.

[14] ONODA, MAURICIO. Estudo sobre um algoritmo de árvores de decisão acoplado a um sistema de banco de dados relacional. Dissertação (Mestrado) – UFRJ. Rio de Janeiro-RJ. 2001.

[15] BREIMAN, LEO. Random forests, 2001. Disponível em: <http://link.springer.com/article/10.1023/A:1010933404324>. Acesso em Maio de 2016.

[16] MCCANNE, STEVEN AND JACOBSON, VAN. The BSD Packet Filter: A New Architecture for User-level Packet Capture. 1993.

[17] GALVÃO, RICARDO K. M. Introdução à análise forense em redes de Computadores: Conceitos, técnicas, ferramentas para “grampos digitais”. São Paulo. Editora: Novatec. 2013.

[18] FILHO, R. H. RIBEIRO, VICTOR P. A. MAIA, JOSÉ E. B. Classificação online dos tráfegos TCP e UDP baseada em sub-fluxos. XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos. Campo Grande-MS. 2011.

[19] RIBEIRO, V. P. FILHO, R. H, and MAIA, JOSE E. B. Online traffic classification based on sub-flows. International Symposium on Integrated Network Management (IEEE). Dublin, Ireland. 2011.

Encadeamento de serviços em rede: uma nova abordagem para provisionamento dinâmico de serviços

Tales Anaximandro do Bonfim Visgueira [1]
Ricardo Gomes Queiroz [1]
Christian Esteve Rothenberg [2]

Resumo: A expansão tecnológica na Internet tem motivado pesquisadores a buscar técnicas no intuito de flexibilizar o comportamento das redes, bem como minimizar custos de operação e prover serviços de forma dinâmica para adequação às novas necessidades tecnológicas. Neste contexto, este artigo apresenta o conceito do encadeamento dinâmico de serviços em redes de computadores, do inglês *Network Service Chaining (NSC)*, exemplificado usando uma implementação protótipo baseada no controlador OpenDayLight.

***Palavras-chave*:** Encadeamento de Serviços em Redes. OpenDayLight. Redes Definidas por Software.

Abstract: *The technological expansion on the Internet has motivated researchers to seek techniques in order to make more flexible the network behavior, and minimize operating costs and provide services to dynamically conform to the new technological requirements. In this context, this paper the concept of Network Service Chaining (NSC) exemplified through a prototype implementation based on the OpenDayLight controller.*

***Keywords*:** *Network Service Chaining. OpenDayLight. Software Defined Networking*

1 INTRODUÇÃO

A crescente demanda na utilização dos meios de telecomunicações tem proporcionado uma explosão de novas soluções tecnológicas (e.g., Telefonia Móvel, Virtualização de Máquinas, Computação em Nuvem e a revolucionária Internet das Coisas), e devido ao desenvolvimento dessas novas soluções, há a previsão de existir mais de 50 bilhões de dispositivos conectados na Internet até 2020 [1]. Durante a realização de pesquisas para evolução da infraestrutura das redes, os pesquisadores identificaram a existência de problemas que tornam difícil a realização de mudanças no núcleo das redes, como a existência de uma grande quantidade de protocolos de comunicação, desenvolvidos por fabricantes de dispositivos diferentes (protocolos proprietários) e a existência de "caixas pretas", implementações verticalmente integradas em plataformas *software* e *hardware* também conhecidos como *middlebox* [2].

Associados a estes problemas a arquitetura atual das redes de computadores, projetada na década de 80 (há mais de 30 anos) e estruturada na comutação de pacotes com a arquitetura TCP/IP, consiste em uma estrutura de rede calcificada ou ossificada (engessada), uma analogia ao processo de envelhecimento das cartilagens dos ossos dos seres vivos quando atingem o seu nível de amadurecimento máximo, e por essa razão, dificultam a implementação de novas soluções tecnológicas [3]. Adicionalmente aos problemas comentados, em ambientes corporativos como *data centers*, núcleos de Computação em Nuvem e soluções para a Internet das Coisas, existe a necessidade de disponibilizar infraestrutura ou serviços de rede para atender às demandas dos usuários, onde normalmente são atendidas com a utilização de Máquinas Virtuais, do inglês *Virtual Marchine (VM)*. A evolução para este novo ambiente virtual proporcionou o surgimento de uma nova geração de redes inteligentes e flexíveis baseadas em software e com uma abordagem revolucionária.

Segundo McKeown et al. (2008), o paradigma das Redes Definidas por Software, do inglês *Software Defined Networking (SDN)*, surgiu como uma abordagem tecnológica visando promover a inovação das redes de

[1]Pós-Graduação Lato Sensu em Redes de Computadores - Faculdade Santo Agostinho (FSA) - Teresina (PI) - Brasil
{talesvisgueira,rgqueiroz@gmail.com}
[2]Departamento de Engenharia, Computação e Automação Industrial - Unicamp - Campinas (SP) - Brasil
{chesteve@dca.fee.unicamp.br }

computadores. O principio básico do paradigma SDN é a capacidade de programar os elementos da rede através de uma interface programática de software (API), que desacopla o plano de dados do plano de controle. Outra tecnologia emergente é a virtualização de funções de redes, do inglês *Network Function Virtualization (NFV)*, que desacopla a função executada por um sistema (*software*) de sua parte física (*hardware*). Apesar dessas inovações trazerem avanços na gerência das redes, existe ainda a necessidade de escalonar os serviços de forma dinâmica e flexível. Atualmente, esses serviços são implantados de forma estática e dependem de configurações fixas que tornam a abordagem inflexível e difícil de adaptar-se às mudanças de requisitos.

Dentro do contexto acima, o problema desta pesquisa consiste em tornar o provisionamento dos serviços nas redes de computadores mais escaláveis e flexíveis. A hipótese em estudo é que a integração dos conceitos de gerenciamento centralizado e programável, somados à capacidade de virtualização dos dispositivos existentes nas tecnologias SDN e NFV, podem otimizar a forma de provisionamento dos serviços. Nesta abordagem, o objetivo principal é implementar o conceito de Encadeamento de Serviços em Redes (NSC), com a finalidade de prover serviços de forma dinâmica através da técnica de Cadeia de Funções de Serviços, do inglês *Service Function Chains (SFC)*.

Os objetivos específicos serão desenvolvidos com os seguintes passos: i) analisar e apresentar os novos conceitos da próxima geração de redes (e.g., SDN, NFV e SFC) aos administradores não familiarizados com essas tecnologias; ii) implementar o estado da arte no provimento de serviços com a técnica SFC, disponibilizada pelo controlador SDN de código aberto OpenDayLight (ODL) e; iii) avaliar a implementação prova de conceito de NSC em um ambiente de experimental baseado no emulador Mininet. A justificativa para a realização deste trabalho baseia-se na necessidade de explorar a tecnologia de NSH no contexto de SDN e NFV como paradigmas na área de redes para disponibilizar, de forma otimizada, políticas e serviços em diferentes ambientes de redes de computadores para atender a crescente demanda de clientes e soluções tecnológicas.

A organização deste trabalho está estruturada da seguinte forma: inicialmente na seção 2 é realizada a revisão teórica referente as tecnologias SDN e NFV, dando sequência, na seção 3 é apresentada a arquitetura do Encadeamento de Serviços em Rede (NSC) para posteriormente, na seção 4 ser discutido o processo de implementação do provisionamento dinâmico de serviços com o controlador OpenDayLight e por fim, na seção 5 são apresentadas as conclusões obtidas no experimento e os aspectos para implantação do conceito NSC em novos ambientes.

2 REFERENCIAL TEÓRICO

Atualmente existem duas abordagens com a finalidade de decidir o "Futuro das Redes". A primeira, denominada **Evolucionária**, argumenta que a arquitetura TCP/IP, elemento chave no modelo das redes atuais tem funcionado satisfatoriamente, não sendo recomendável uma ruptura completa, apenas a evolução do próprio protocolo. A segunda abordagem tem uma visão mais **Revolucionária** e defende a criação de uma nova rede, deve-se abandonar o modelo atual e desenvolver uma arquitetura inteiramente nova [5]. Essa nova ideia de recomeçar do zero é conhecida entre os pesquisadores como *Clean-Slate* (algo como "Tela em Branco"), ou arquitetura disruptiva [6]. No entanto, apesar de existirem ideias distintas, há um consenso entre os pesquisadores em relação à necessidade de realizar mudanças estruturais nas redes para torná-las mais eficientes e flexíveis.

No contexto das abordagens evolutivas, foram desenvolvidas soluções emergenciais (remendos) para minimizar os efeitos ocasionados pela crescente expansão da arquitetura das redes, destacando-se as seguintes: i) *Classless Internet Domain Routing (CIDR)*, uma solução definida para estender o endereçamento IPv4; ii) *Network Address Translation (NAT)*, uma solução de interconexão entre redes privadas e públicas; iii) *Dynamic Host Configuration Protocol (DHCP)*, um serviço para distribuição de endereços IP e informações adicionais da rede (e.g., máscara de sub-rede, gateway, endereço de DNS etc). Já no contexto revolucionário, a nova geração de IP conhecida como IPv6 e definida na RFC 1550:1993 como um novo protocolo para transporte de dados, iniciou o processo de mudanças *Clean-Slate*. Uma de suas principais características inovadora é o aumento do endereçamento IP de 2^{32} para 2^{128}, o equivalente a 340 undecilhões de novos endereços. Outras soluções revolucionárias também estão em fase de implantação (e.g., redes programáveis e funções de rede virtualizadas) [5].

Na implementação dessas novas soluções, os pesquisadores sempre esbarravam em dificuldades de realizar mudanças nas redes, pelo fato dos trabalhos de pesquisas permanecerem em projetos de laboratórios e as redes atu-

ais serem constituídas por dispositivos proprietários (roteadores/*switches*) contendo componentes internos também chamados de *middlebox* ou "Caixas Pretas" normalmente implementados por fornecedores distintos (e.g., Cisco, Hewlett-Packard, Juniper, NEC), que tornam as mudança difíceis de serem testadas em grande escala.

A Força Tarefa de Engenheiros da Internet, do inglês *Internet Engineering Task Force (IETF)*, define na RFC-3224:2002, o termo *Middlebox* como qualquer dispositivo intermediário que executa funções específicas no encaminhamento de datagramas entre um host de origem e o destino [7]. Esses componentes desempenham funções bem específicas como Firewall, NAT, Proxy, Sistemas de Detecção de Instrução (IDS/IPS), *Load Balancer*, etc. Embora estas ferramentas ofereçam um ponto de vista central de administração, operam no nível de protocolos e necessitam de engenheiros de redes para configurar políticas definidas em alto nível para comandos de baixo nível. Esta forma de operação, diminui a inovação e aumenta os custos operacionais de gestão das redes.

2.1 Origem das redes programáveis

Com a finalidade de tornar as redes atuais mais flexíveis e gerenciáveis, com base na analogia da relativa facilidade de programar os computadores pessoais, os pesquisadores propuseram soluções que contribuíram na formação do conceito de redes programáveis, onde destacam-se as seguintes: Redes Ativas, NetFPGA, ForCES, SANE, Ethane e o protocolo OpenFlow.

- **Redes Ativas**, do inglês *Active Networking*, foi uma solução proposta em 1990 como uma abordagem que utiliza uma interface de programação (API) para expor os recursos de redes (e.g., filas de processamento, armazenamento e pacotes), apoiados em aplicações com funcionalidade personalizada para aplicar regras em um subconjunto de pacotes que passam através do nó, semelhante a um sistema operacional de rede em cada nó [8]. Esta abordagem foi considerada inadequada, pois havia o entendimento que a simplicidade no núcleo das redes era fundamental para o sucesso da Internet. Neste sentido, as Redes Ativas foram uma das primeiras abordagens *Clean-Slate* que exploravam os serviços prestados pela pilha tradicional da Internet.

- A solução **NetFPGA** foi iniciada no final de 2001 na Universidade de Stanford, como uma plataforma aberta que combina memórias (SRAM e DRAM), processadores de sinais e interfaces de redes Ethernet (1 Gbps ou 10 Gbps) em uma placa padrão PCI, denominada *Field-Programmable Gate Array - FPGA*. Sua finalidade é permitir aos desenvolvedores programar o FPGA para implementar novos mecanismos, protocolos e arquiteturas. A NetFPGA possibilita modificar os pacotes em trânsito e controlar o encaminhamento do tráfego de rede. Segundo [9], o projeto NetFPGA foi motivado pela necessidade de criar ferramentas para ensinar sistemas de rede de computadores em nível de graduação e pós-graduação.

- A solução **ForCES**, do inglês *Forwarding and Control Element Separation* é considerada a primeira iniciativa desenvolvida para separação do elemento de controle e encaminhamento, sua proposta foi apresentada inicialmente pela IETF com a RFC 3654 em 2003. A ideia era redefinir a arquitetura interna dos dispositivos de rede, separando os elementos de controle e encaminhamento [10]. No entanto, o dispositivo de rede ainda é representado como uma entidade única.

- O projeto **SANE**, do inglês *Secure Architecture for the Networked Enterprise (SANE)* foi concebido como uma arquitetura segura para redes empresariais em 2006, sua finalidade era fornecer uma única camada de proteção para redes, localizada entre as camadas Ethernet e de Rede, similar ao que ocorre com as *Virtual Local Area Network (VLANs)*. Nessa abordagem rotas são construídas por um controlador de domínio logicamente centralizado, que concede acesso usando um ponto de vista global, permitindo implementar políticas de uma forma independente de topologia [11]. Em contraste com as políticas de redes existentes, que baseiam-se em regras nos firewalls e outros middleboxes com dependências implícitas sobre como a topologia das redes estão implantadas .

- O projeto **Ethane** (sucessor do SANE) estabelece uma solução logicamente centralizada a nível de fluxo para o controle de acesso em redes empresariais. Sua principal característica era reduzir as interrupções às tabelas de fluxo nos dispositivos de comutação de dados. Essas tabelas são preenchidas por um controlador central, que utilizava políticas de segurança de alto nível. O projeto foi implementado inicialmente no departamento de ciência da computação de Stanford em 2007, uma preparação do terreno para a criação do protocolo OpenFlow [12].

- O protocolo **OpenFlow** foi desenvolvido em 2008 como uma solução baseada na abordagem *Clean-Slate*, considerada a que mais teve sucesso na implementação da separação dos planos de controle e dados. Sua principal finalidade é controlar o fluxo de dados, definir as rotas que os pacotes vão seguir e o processamento que devem ser submetidos.

Todas essas soluções colaboraram para o surgimento de um novo paradigma de flexibilização e otimização, chamado de **Redes Definidas por Software (SDN)** [8]. O termo SDN teve sua proposta inicial na publicação do artigo de Greene (2009), que relatou os trabalhos dos pesquisadores Nick McKeown, Martin Casado e outros membros da Universidade de Stanford, motivados pela incapacidade de mexer nos dispositivos de encaminhamento, e por este motivo, desenvolveram um novo padrão que acessa as tabelas de fluxos e modificam o comportamento de encaminhamento dos pacotes da rede [8]. Atualmente, o órgão responsável para evoluir o conceito SDN é a fundação *Open Networking Foundation - ONF*, estruturada em vários grupos de trabalhos (GTs) e focados em promover às áreas de Extensibilidade, Transporte Óptico, Rede Móvel e sem Fio. A IETF também definiu o conceito SDN, através da RFC-71409:2014, como um conjunto de técnicas utilizadas para facilitar o projeto, entrega e gestão de serviços de rede de uma maneira determinista, dinâmica e escalável [13].

2.2 Redes Definidas por Software (SDN)

A arquitetura clássica de SDN é dividida em três planos: plano de aplicação, controle e dados. No plano de aplicação encontram-se os aplicativos (softwares especialistas) que podem receber eventos de redes e atuar de forma proativa ou reativa para oferecer serviços ou configurações personalizadas. No plano de controle, fica o controlador central, também conhecido como Servidor Operacional de Rede, do inglês *Network Operating System (NOS)*. Já o plano de dados é constituído por dispositivos (Roteadores ou Switchs) que realizam a função de encaminhar os dados. A Figura 1 ilustra um modelo da arquitetural SDN.

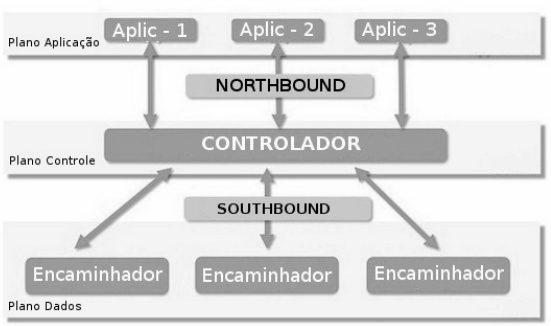


Figura 1: Adaptação do modelo da arquitetura SDN
Fonte Intel Digital Labs, 2014

Aplicações SDN são soluções a nível de software implementas em alguma linguagem de programação (e.g,. C, C++, Java, Python, etc.) com a finalidade de processar as mensagens enviadas pelo plano de dados ao controlador e repassada às aplicações através de uma API disponibilizada pelo próprio controlador SDN. Essa característica possibilita uma capacidade programável para tomada de decisão sobre quais ações devem ser realizadas pelo dispositivo de encaminhamento, por este motivo as aplicações podem monitorar o estado dos fluxos de dados e permitir alteração no comportamento da rede.

A Northbound API (interface para o norte, em português), representa uma interface de comunicação entre o plano de controle e às aplicações SDN para permitir o gerenciamento sobre as funções de rede. A função principal da API é traduzir os requisitos das aplicações de gerenciamento em instruções de baixo nível para os dispositivos da rede e transmitir estatísticas geradas nos dispositivos de encaminhamento e processadas pelo controlador para às aplicações SDN. Essa interface abstrai os eventos e conjuntos de instruções de baixo nível dos dispositivos de comutação para possibilitar às aplicações acopladas ao controlador decidir quais ações de encaminhamento, encapsulamento ou descarte (drop) de pacotes serão realizadas. Normalmente a API é implementada por um dos protocolos a seguir: Flow Management Language (FML), Procera, Frenetic, RCP ou REST.

A Interface Southbound (interface para o sul, em português), representa o elo de ligação entre o plano de controle e os elementos de encaminhamento de dados, sendo, portanto, o elemento crucial para separar claramente a funcionalidade de controle e plano de dados. Essa API está fortemente ligada aos elementos de encaminhamento da infraestrutura física ou virtual, podendo ser implementada pelos protocolos OpenFlow e ForCES, Protocol-Oblivious Forwarding (POF), Path Computation Element (PCE) e Interface to the Routing System (IRS)). Suas principais funções são i) gerar eventos com mensagens sobre o comportamento da rede (e.g., mudança de porta ou enlace); ii) enviar estatísticas do fluxo de dados e características da rede e; iii) enviar pacotes do fluxo de dados para o controlador com a finalidade de decidir qual o comportamento de encaminhamento.

Segundo Costa (2013), o elemento central na arquitetura SDN é o controlador, responsável por disponibilizar um ambiente programático de controle onde os desenvolvedores podem ter acesso aos eventos gerados por uma interface de rede e gerar comandos para controlar o fluxo de dados nos dispositivos de encaminhamento através de suas interfaces programáticas NorthBound ou SouthBound [2]. As interfaces programáticas (API) possibilitam a implementação de políticas baseados em níveis de abstrações maiores, com a centralização da inteligência e lógica de roteamento no plano de controle onde estão armazenadas as tabelas de Bases de Informações de Roteamento, do inglês *Routing Information Base (RIB)*. Essas informações centralizadas permitem a realização da comparação dos fluxos de dados e a inclusão ou remoção de regras nas tabelas de encaminhamento, denominadas de *Forwarding Information Base (FIB)* localizadas nos equipamentos de comutação [14]. O primeiro controlador construido para o contexto SDN foi o NOX que motivou a implementação de vários outros controladores, a Tabela 1 lista os principais controladores de código livre disponíveis na comunidade atualmente.

Tabela 1: Principais controladores SDN open source.

NOME	ANO	LINGUAGEM	PLATAFORMA
NOX	2008	C++	Linux
POX	2010	Python	Windows, Mac e Linux
Maestro	2010	Java	Windows, Mac e Linux
Beacon	2010	Java	Windows, Mac, Linux e Android
RYU	2011	Python	Windows, Mac, Linux
FloodLight	2011	Java	Windows, Mac, Linux
OpenContail	2013	Python	Windows, Mac, Linux
OpenDayLight	2014	Java	Linux
ONOS	2014	Java	Windows, Mac, Linux

Outro importante componente da arquitetura SDN é o Agente, um *firmware* (software embutido no hardware) que deve ser instalado nos equipamentos de encaminhamento (EE) para habilitar a arquitetura SDN. A instalação desse software, possibilita aos EE's utilizarem um protocolo de comunicação para intermediar as mensagens entre os planos de dados e controle. As tabelas de fluxo de dados existentes nos EE's são formadas por estruturas com um conjunto de entradas (regras), junto a campos de cabeçalhos (*header*), instruções (ações) e campos estatísticos (número de bytes, duração do fluxo etc), definidos para manusear os pacotes do tráfego de dados [15], conforme Figura 2.

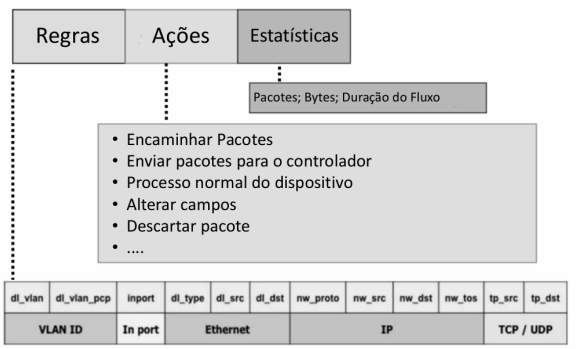


Figura 2: Definição de um fluxo na arquitetura OpenFlow
Fonte SBRC Tutorial OpenFlow, 2010

A definição de fluxo é representada pelo conjunto de ações que formam uma entrada da tabela de fluxos, como ilustrado na Figura 3. Os EE's quando em operação, analisam cada pacote que chega e comparam os cabeçalhos dos pacotes com as entradas das tabela de fluxos. Caso seja encontrado um casamento, considera-se que o pacote pertence àquele fluxo, então são aplicadas as ações existentes ou se por acaso, nenhuma regra for encontrada, o pacote é encaminhado para o controlador. A comunicação entre os EE (comutadores) e o controlador é determinada por um dos protocolos de comunicação da Southbound, que definem as regras que serão aplicadas aos fluxos nos dispositivos de comutação. Essa comunicação ocorre em um canal seguro de rede e pode empregar criptografia (SSL ou TLS) para garantir o sigilo no tráfego dos dados.

PORT	VLAN	MAC Src	MAC Dst	Eth Type	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Ação
*	*	*	00:1f ...	*	*	*	*	*	*	Porta 6
p3	vl1	00:20 ...	00:1f ...	0800	1.2.3.4	5.6.7.8	4	1765	80	Porta 6
*	*	*	*	*	*	*	*	*	22	Drop
*	*	*	*	*	*	5.6.7.8	*	*	*	Porta 6
*	vl1	00:1f. ..	*	*	*	*	*	*		Portas 6,7,8

Figura 3: Tabela de fluxo da arquitetura OpenFlow.

Dentro do contexto da SDN, o controlador realiza a verificação de sua tabela de fluxo e a notificação das aplicações acopladas ao controlador através da geração de eventos, que podem resultar na criação de uma nova entrada no comutador, fazendo assim com que os próximos pacotes, contendo o mesmo cabeçalho, sejam encaminhados pelo próprio comutador. Isto evita o reenvio dos pacotes para o controlador novamente, resultando em ganho de desempenho. Existem cinco ações básicas associadas a cada entrada na tabela de fluxos de um comutador programável: i) encaminhar os pacotes deste fluxo para uma determinada porta (ou portas); ii) encapsular e transmitir pacotes deste fluxo para um controlador e; iii) descartar pacotes deste fluxo; iv) alterar o conteúdo dos campos e; v) executar o processo normal do EE. Estas ações são realizadas através de trocas de mensagens de controle entre o controlador e o switch, de forma assíncronas e simétricas.

2.3 OpenDaylight (ODL)

O ODL é um projeto desenvolvido e apoiado pela *The Linux Foundation*, cujo objetivo é acelerar adaptação da tecnologia SDN através de uma plataforma modular e extensível. Essa organização é apoiada por várias empresas que comprometeram-se em prover ajuda economica e recursos técnicos ao desenvolvimento da plataforma (e.g., Arista Networks, Big Switch Networks, Brocade, Cisco, Citrix, Ericsson, HP, IBM, Juniper Networks, Microsoft, NEC, Nuage Networks, PLUMgrid, Red Hat and VMware). Além dos motivos comentados, o controlador ODL utiliza em sua arquitetura o *framework Open Services Gateway Initiative (OSGi)*, uma solução que permite a implantação de aplicações no formato de plugins (módulos) e disponibiliza a funcionalidade de encadeamento de serviço (SFC) através da API Northbound com o protocolo REST, acrônimo de *Representational State Transfer (REST)*, uma arquitetura para construir web services escaláveis através do protocolo HTTP. Essas características, possibilitam operadores de rede criar, atualizar e excluir cadeias de serviços, bem como especificar a troca de metadados entre os nós de um caminho de serviço através do uso de uma interface web.

Adicionalmente às características já comentadas, o controlador implementa uma camada de abstração de serviço, do inglês *Service Abstration Layer (SAL)*, que permite a comunicação entre os plugins através de mensagens de exportação que ignora o papel das APIs *Southbound* e *Northbound*, baseando-se na definição de plugins consumidor e provedor. Esta alteração significa que cada plugin dentro do ODL pode ser visto como um fornecedor ou consumidor, podendo trocar informações apenas pelo fluxo de mensagens entre os encaixes envolvidos. Os provedores são plugins que expõem recursos para aplicações e outros plugins através da sua API *Northbound*, os consumidores são componentes que fazem uso dos recursos fornecidos por um ou mais provedores.

A funcionalidade principal da SAL é descobrir como um serviço solicitado vai ser executado, independentemente do protocolo subjacente usado entre o controlador e os dispositivos de rede. Neste sentido, a realização do

controle dos dispositivos em seu domínio passa a saber apenas quais dispositivos estão na rede, suas capacidades e forma de acessibilidade, cujas informações são coletadas, armazenadas e gerenciadas pelo gerenciador de topologia. Com esta abordagem, os módulos da API Southbound ficam ligados dinamicamente na camada da SAL, possibilitando suporte a vários protocolos (e.g., OpenFlow 1.0, OpenFlow 1.3, OpenVSwitch, NetConf com Yang e BGP-LS). Essa arquitetura apresenta duas diferentes abordagens: *API-Driven SAL (AD-SAL)* e o *Model-Driven SAL (MD-SAL)*.

A abordagem AD-SAL realiza ações reativas através do recebimento de eventos da rede, às aplicações são acopladas ao controlador com o framework OSGi e o comportamento padrão é Stateless (não guarda estado). Já a abordagem MD-SAL possui uma programação proativa, sem a possibilidade de recebimento direto de eventos da rede e possui um modelo agnóstico que suporta apenas alguns dispositivos e modelos. Além disso o modelo de dados de estruturas das cadeias de servicos são definidas em arquivos e manipulados por linguagens de modelagem como por exemplo a linguagem YANG. Esta linguagem define a sintaxe e semântica na modelagem da camada de conteúdo sendo compatível com o modelo *Structure of Management Information (SMIv2)*.

2.4 Virtualização de Funções de Rede (NFV)

O grupo que lidera as especificacoes NFV é o *European Telecommunications Standards Institute (ETSI)*, através do grupo de trabalho *Industry Specifications Group (ISG)*, que trabalha para mover os serviços dos servidores para equipamentos comuns ou virtualizados. As motivações para a implantação dessa tecnologia são: i) reduzir os custos dos equipamentos e o consumo de energia, através da redução da quantidade de equipamento físicos para a disponibilização de serviços; ii) aumentar a velocidade de instanciação e disponibilização de novos serviços e; iii) virtualização dos recursos da rede de modo a atender um maior número de clientes.

Essa tecnologia permite criar funções de redes virtualizadas, do inglês *Virtualization Network Function (VNF)*, uma combinação de serviços dentro de um servidor de uso geral que disponibiliza funções de rede, do inglês *Network Function (NF)*. O termo NF, representa um conjunto de blocos funcionais dentro de uma infraestrutura de rede com comportamento bem definido [12]. A RFC-7665:2015 define o termo NF, como um nó de rede que tem a função de realizar o tratamento específico de pacotes recebidos (e.g., *Firewall, Dynamic Host Configuration Protocol (DHCP), Network Address Translation (NAT), Intrusion Detection System (IDS), Deep Packet Inspection DPI* etc).

As VNF, são implementadas dentro de uma Infraestrutura de Virtualização de Funções de Rede (NFVI), que inclui uma diversidade de recursos físicos para suportar a execução das NF. A camada de gerenciamento e Orquestração das NFV, do inglês *Network Function Virtualization Management and Orchestration (NFV-MANO)* é responsável por controlar o ciclo de vida e gestão física dos recursos. A arquitetura do NFVI possui três domínios: i) **Computacional**, inclui recursos de computação para VNFs através da camada de virtualização armazenados em storages (e.g., NAS, SAN); ii) **Virtualização**, camada que abstrai os recursos de hardware e dissocia o VNF do hardware. Normalmente, este tipo de funcionalidade é fornecida por hypervisors. iii) **Infra-estrutura de rede**, um domínio que inclui tanto os recursos de redes físicos e virtuais para interligar os recursos de computação dentro da NFVI, visualizar Figura 4.

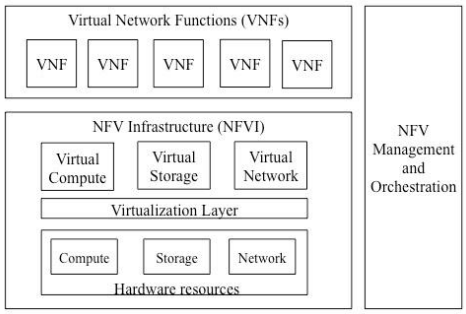


Figura 4: Arquitetura Network Function Virtualization (NFV)
Fonte ETSI ISG 4G Americas, 2014.

2.5 Trabalhos relacionados

John et al. (2013) abordaram em seu artigo as vantagens da adaptação do conceito NSC e as direções de pesquisa para o encadeamento de serviço. Detalham também os principais aspectos a serem considerados durante o ciclo de vida típico de uma cadeia de serviços, no contexto das redes de telecomunicações [4]

Os pesquisadores Li and Chen (2015) apresentaram uma investigação aprofundada sobre o desenvolvimento de NFV e sob a arquitetura NFV-Definida por Software, com ênfase no encadeamento de serviço como uma aplicação. Primeiro, introduziram a arquitetura NFV definida por software como o estado da arte e as relações atuais entre NFV e SDN. Com isso, fornecem uma visão histórica do envolvimento de middlebox com NFV, além de apresentar os desafios mais significativos e soluções relevantes para a NFV [16].

Brown (2015) apresentou uma avaliação sobre o porque cadeias de serviço são úteis e como operadoras podem habilitar a criação de serviços dinamicamente com a utilização de equipamentos dedicados (appliance) e funções de redes virtualizadas. Em sua análise ele aplicou várias opções de implementação de cadeias de serviço, abordando especificamente o papel do classificador, opções de cabeçalho do pacote e a importância dos metadados [17].

Ruckert (2015) implementou uma prova de conceito com uma abordagem de encadeamento de serviço totalmente baseada em software para cadeias de serviço. O protótipo é executado no padrão hardware e inclui um conjunto de funções de rede como exemplo para mostrar a exequibilidade da abordagem. Além disso, ele utiliza uma API de alto nível para integrar um operador e um portal de auto-serviço ao cliente [18].

3 ENCADEAMENTO DE SERVIÇOS EM REDE (NSC)

Hoje a disponibilização de serviços nas redes de computadores exige a configuração e instanciação de funções de rede manualmente, essa demanda envolve a intervenção humana e por este motivo está propenso a ocorrência de erros e a uma maior carga de trabalho pela grande quantidade de dispositivos que necessitam ser configurados. A iniciativa de integrar as tecnologias SDN e NFV permite aproveitar o que há de melhor em ambas arquiteturas para configuração e reconfiguração a rede através de um controlador central, bem como a utilização de toda a capacidade de virtualização de máquinas e serviços, como pode ser visto na Figura 5.

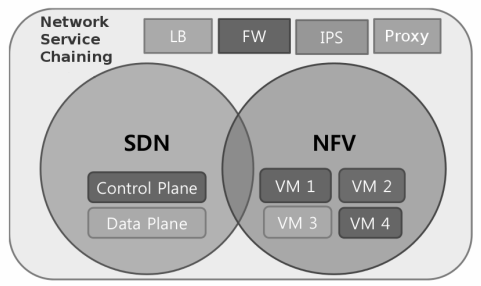


Figura 5: Integração das arquiteturas SDN e NFV para o conceito NSC

A integração das tecnologias SDN e NFV possibilita a utilização de uma nova abordagem conhecida como Encadeamento de Serviços em Rede, do inglês *Network Service Chaining (NSC)*, que implementa uma abstração em nível de plano de serviço ao lado do plano de controle para direcionar o fluxo de dados dinamicamente através dos gráficos de serviços criados. Segundo John et al. (2013), o encadeamento dinâmico de serviço é definido como um processo de entrega contínua de serviços com base em associações de função de rede. Neste contexto, a entrega contínua significa uma orquestração dinâmica das funções de rede para a implantação automática de melhorias na eficiência operacional, recuperação de falhas, bem como na capacidade da realização de testes [4].

3.1 Cadeias de Função de Serviço (SFC)

Como discutido nas seções 2.2 e 2.3, a integração das soluções existentes na nova geração das redes (SDN e NFV), habilita a criação de uma nova técnica para criação de Cadeias de Funções de Serviço, do inglês *Service Function Chains (SFC)*, com a finalidade de criar topologias em um gráfico de caminhos contendo funções de serviços em rede, como pode ser visto na Figura 6.

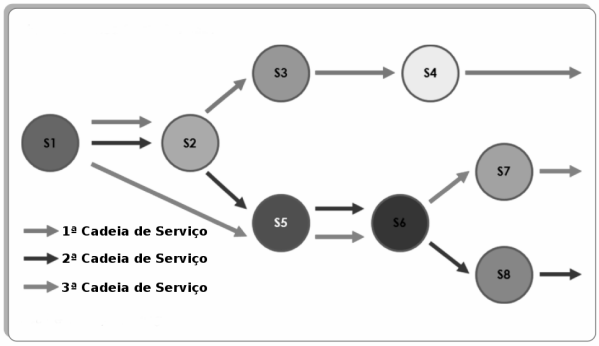


Figura 6: Exemplo de gráfico com funções de serviço
Fonte Heavy Reading, 2014

A RFC-7665:2015 define SFC como um conjunto ordenado de abstrações de Funções de Serviços (SF) que devem ser aplicadas aos fluxos de pacotes selecionados como resultado de uma classificação. Em suma, são funções virtuais do tipo LoadBalancer, Firewall, IDS/IPS, NAT, QoS, Proxy, DPI etc [19].

Cada uma dessas SFs podem ser realizadas através de um serviço dedicado e são atravessadas em uma dada ordem (1). A conectividade entre as SFs podem ser representadas por um gráfico conhecido como Gráfico de Função de Rede, do inglês *Network Function Graphic (NFG)*, sendo definido em uma formulação matemática de decomposição dos serviços, representada por um mapeamento de cada NF em um conjunto de NFGs (2).

$$Classificador \longrightarrow SFF \longrightarrow S1 \longrightarrow S2 \longrightarrow S3 \longrightarrow S4 \tag{1}$$

$$NFi \longrightarrow \{NFGi1, NFGi2, ...\} \tag{2}$$

Os principais componentes da SFC são o classificador, o plano de controle e os encaminhadores de funções de serviço. O classificador, do inglês *Classifier*, possui habilidade para identificar e classificar o tráfego antes de encaminhar para processamento pelo gráfico de serviço. O plano de controle é responsável por definir a topologia, políticas e o caminho do fluxo pelo gráfico de serviço e os Encaminhadores de Funções de Serviço, do inglês *Service Function Forwarder (SFF)*, tem a função de determinar o destino do tráfego.

Nessa arquitetura um Controlador SDN (*Controller*) gerencia o Classificador e os SFF através do Plano de Controle para atualizar as tabelas de fluxo dos EE. Neste sentido, o classificador (*Classifier*) é colocado no início do tráfego e define as cadeia de serviços com a inclusão de uma etiqueta no cabeçalho do pacote chamado de Cabeçalho de Serviço de Rede, do inglês *Network Service Header (NSH)*, essa etiqueta determina o tipo de serviço a ser aplicado ao fluxo. O "Serviço"diz respeito ao tipo de SF e à ordem através da qual o fluxo passa por elas, neste sentido a implementação de cadeias de serviços pode ser realizada a partir de um ou mais serviço das camadas 4 a 7 do modelo OSI.

As cadeias de serviços são identificadas por um identificador (ID) único e sua estrutura é alocada em um plano de serviço para posterior consulta pelos SFF. Novas funções de serviço, baseadas em políticas e metadados de negócio existente no plano de controle podem ser adicionadas, removidas ou alteradas quando necessário, basta simplesmente mudar a etiqueta para obter uma nova ramificação da cadeia de acordo com os resultados do processamento da função de serviço [17]. Os SFF representam instâncias virtuais colocados na rede para analisar a etiqueta e determinar o destino do fluxo. Sua finalidade é fornecer serviço de transporte às cadeias de serviços. Neste ponto, os pacotes do fluxos de dados são encapsulado e enviados através de uma rede virtual dedicada a esse conjunto de funções de serviço. A Figura 7 exibe o fluxo da SFC.

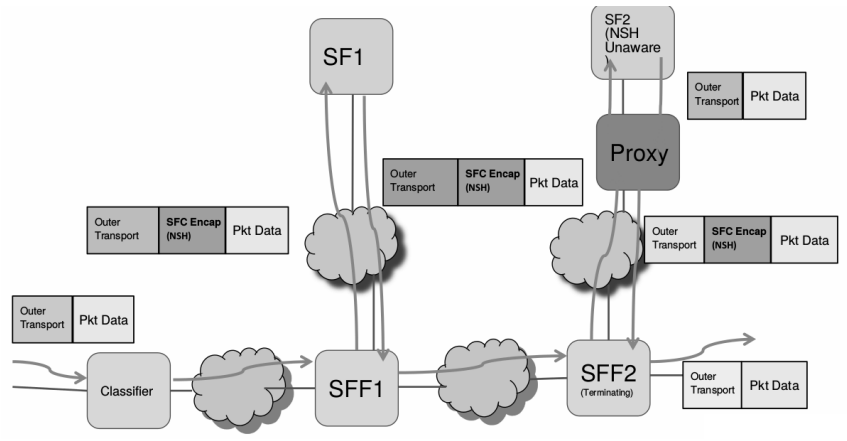


Figura 7: Encadeamento de Funções de Serviço (SFC)
Fonte OpenDayLight.org, 2015

3.2 Encapsulamento NSH

O encaminhamento dos pacotes pelo gráfico de funções de serviço é realizado com a inclusão de uma etiqueta no cabeçalho do pacote que identifica os tipos de funções de serviço e a ordem que os pacotes são aplicadas ao fluxo, sendo os pacotes transferidos com base nesta etiqueta. No documento-rascunho DRAF-SFC-NSH-04, de 13 de março de 2016, é proposto o protocolo NSH como a forma padrão de encapsulação da técnica SFC, o documento descreve com detalhe o formato do cabeçalho a ser inserido nos pacotes ou quadros do fluxo de dados para encapsular caminhos de Funções de Serviço, além de detalhar o mecanismo para troca de metadados ao longo dos serviços instanciados. A documentação apresenta a criação de um plano de serviço com configurações e topologias de caminhos de serviço independentes para transportar os pacotes encaminhados sem alterar a topologia da rede principal [20].

Dessa forma a formatação dos campos do protocolo NSH foram divididas em 3 seções: i) base do cabeçalho, fornece informações sobre o cabeçalho do serviço e o protocolo de carga útil (payload); ii) cabeçalho de caminho do serviço, representa a identificação e localização dentro do caminho de serviço e; iii) cabeçalhos de contexto, carregam informação dos metadados e do comprimento variável das informações codificadas, como pode ser visto na Figura 8.

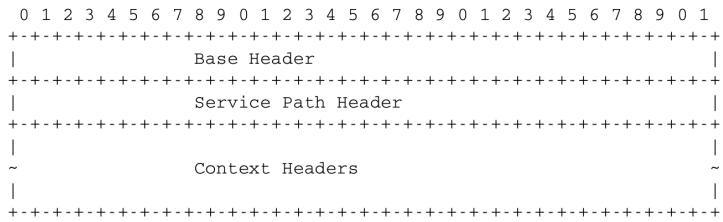


Figura 8: Seções do Network Service Header (NSH),
Fonte: IETF Draf-sfc-nsh-04, 2016

- O base do cabeçalho possui o tamanho padrão de 04 (quatro) bytes, o que corresponde a 32 bits contendo os seguintes campos: i) 02 bits para a versão do protocolo; ii) 08 bits com flags opcionais, do tipo TLVs.; iii) 06 bits para representar o tamanho total do cabeçalho NSH; iv) 08 bits representando o tipo de domínio mandatorio; v) 08 bits com a indicação do próximo protocolo.

- O cabeçalho de caminho do serviço também possui o tamanho de 04 (quatro) bytes com os seguintes campos: i) 24 bits com o identificador para seleção de caminhos de serviço (SPI); ii) 08 bits para armazenar o index do serviço.

- Os cabeçalhos de contexto possui tamanho variável dependendo do valor do campo MD-type da seção do cabeçalho base: i) se o valor for 0x1, o formato terá 4 seções de 32 bits cada contendo informações dos contextos e; ii) se o valor for 0x2, o tamanho do cabeçalho será variável podendo ser zero ou mais cabeçalhos.

Os 02 primeiros bits do cabeçalho base identificam a versão do cabeçalho, os 08 bits de Flags (sinalizadores) seguintes são usados para definir as alterações de cabeçalho ou comportamento de análise. Até o momento, dois sinalizadores estão definidos e os seis restantes estão reservados: i) o bit O: quando definido representa que o pacote é tipo de operação, administração e gestão (OAM); ii) o bit C: indica a presença de um metadados crítico TLV. Os próximos 06 bits representam o tamanho total do cabeçalho NSH (Length).

O campo MD Type é composto por 08 bits e define o tipo de cabeçalho de contexto que armazena a seção de Cabeçalho de Contexto. Se possuir o valor 0x1 o cabeçalho de contexto será configurado com 4 cabeçalhos de 04 bytes cada. Caso o valor seja 0x2, nenhum ou cabeçalho de contexto de tamanho variado pode ser adicionado. O último campo do cabeçalho base é o próximo protocolo que também possui 08 bits com a indicação do tipo do próximo protocolo do pacote original. Existem 03 tipos definidos como 0x1 para IPV4, 0x2 para IPv6 e 0x3 para Ethernet, a Figura 9 abaixo ilustra o formado do NSH.

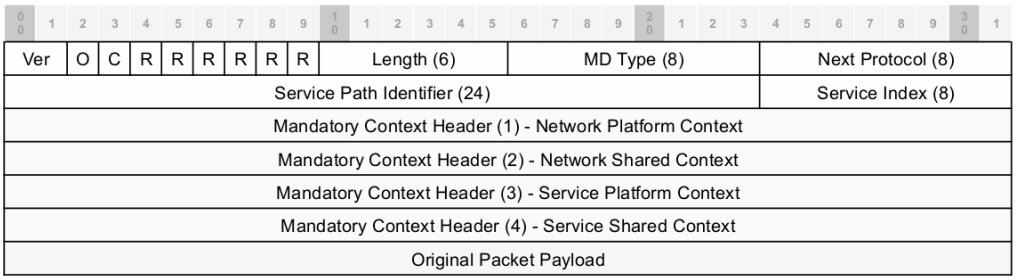


Figura 9: Formato detalhado do Network Service Header (NSH),
Fonte: Cisco Corporation, 2016

Na seção do caminho do serviço o campo Identificador de Caminho do Serviço, do inglês *Service Path Identifier (SPI)*, possui 24 bits e identifica o caminho que interconecta a função de serviço. Este prover um nível de abstração seguindo os nós para selecionar o protocolo de transporte de rede apropriado e a técnica de encaminhamento. O próximo campo desta seção é o Índice de Serviço (SI) com 08 bits, que trata da localização do pacote dentro do caminho de serviço, e quando este é atravessado o SI é decrementado. A combinação do SPI com o SI prover uma clara visibilidade de onde o pacote está dentro do caminho de serviço.

Os cabeçalhos de contextos são constituídos de 16 bytes (128 bits) obrigatórios, definidos como campos de cabeçalhos de contexto para armazenamento de informações de metadados entre os nós de classificação, funções de serviço e os nós da rede. A informação de classificação é exportada dentro do cabeçalho NSH para prover simplicidade de implementação, políticas de funções de serviço e informações de contexto. Os metadados refletem o resultado de uma classificação externa ou antecedente, essa informação de classificação é então provida via NSH por campos do contexto participante da política de decisão local contendo contexto da plataforma e compartilhamento de rede, além do contexto de plataforma e compartilhamento do serviço.

Definida a estrutura do cabeçalho NSH, o próximo passo é a realização do transporte dentro do contexto do plano de serviço, nessa ação pode ser usado qualquer método padrão da indústria que implemente a virtualização de rede, como o *Virtual Extensible LAN (VxLAN)*, Ethernet, *Generic Protocol Extension (GPE)*, *Generic Routing Encapsulation (GRE)* ou a combinação de mais de uma técnica (VxLAN-GPE).

4 IMPLEMENTAÇÃO DO PROTÓTIPO DE ENCADEAMENTO DE SERVIÇO

Nesta seção será demonstrada a implementação do conceito de encadeamento dinâmico de funções de serviço em redes de computadores através da emulação de um cenário virtual como caso de teste. Os softwares selecionados para a implementação foram: a distribuição ODL-SFC e o emulador de rede MiniNet, ferramentas

que possibilitam criar, interagir e customizar protótipos de forma rápida e programável, além de permitirem a realização de testes para depuração e resoluções de problemas.

A distribuição selecionada do projeto ODL é a Beryllium SR2 versão 0.42, com suporte a SFC. Essa distribuição é forneceida dentro de um container chamado de "Karaf", um projeto da fundação Apache baseado em um dos framework OSGi (Equinox ou Felix) para prover características adicionais aos containers. O pacote de distribuição do ODL vem inicialmente com o núcleo principal do controlador e novas características são instaladas sob demanda dentro do container como plugins extensíveis. O processo de instalação das características necessárias para o expererimento deste trabalho consiste na inclusão dos seguintes módulos do projeto ODL: old-restconf; old-mdsal-clustering; old-dlux-core; old-dlux-node; old-dlux-yangui; old-dlux-yangvisualizer; old-l2switch-switch; old-sfc-sb-restt e; old-sfc-ovs.

O Mininet foi utilizado para emular os links, hosts, switches e controladores, reproduzindo processos que executam em espaços de nomes da rede (network namespaces), tendo como principais características: i) fornecer uma maneira simples para a realização de testes em redes com o OpenFlow; ii) permitir a realização múltiplas pesquisas de forma independente; iii) realizar teste em uma topologia grande e complexa, sem a necessidade de uma rede física; iv) incluir ferramentas para depurar e executar testes em toda a rede e v) suportar inúmeras topologias.

O caso de teste para avaliação foi modelado inicialmente com a feramenta CORE e posteriomente implementada no emudador Mininet. A topologia da rede é constituída de equipamentos existentes em três órgãos administrativos na cidade de Teresina, interligados por uma rede metropolitana sem fio, baseada no padrão IEEE 802.16, do inglês *Worldwide Interoperability for Microwave Access (WiMAX)* e conectados a uma rede de fibra óptica no ponto de presença da operadora (Intranet), que estabelece um canal isolado até o ponto de presença na cidade de Fortaleza, como pode ser visualizado na Figura 10.

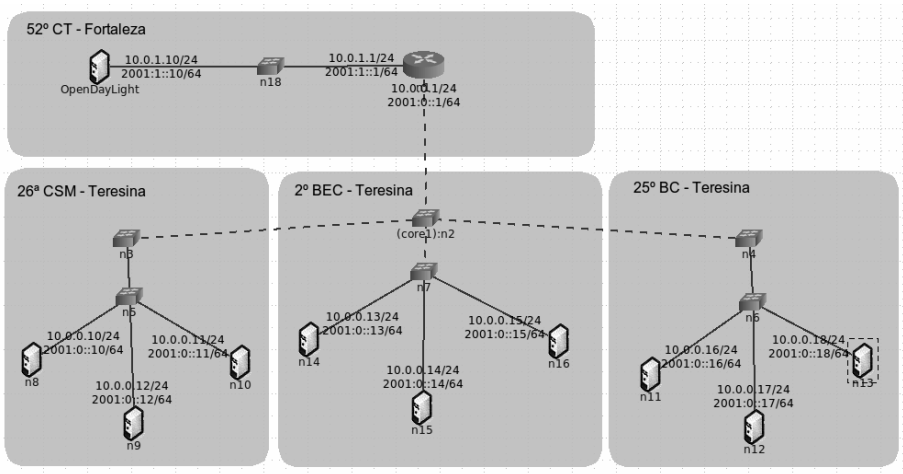


Figura 10: Contexto do caso de teste, autoria própria

A finalidade do caso de teste consiste em criar cadeias de serviços remotamente e de forma dinâmica nos dispositivos de encaminhamento de dados das unidades envovidas, e assim, direcionar o fluxo de dados aos diferentes tipos de serviços de acordo com as requisições das estações clientes existentes ao longo da rede, principalmente ao fluxo destinado aos servidores de conteúdo. As principais ações a serem implementadas são: i) realizar a comutação dos dados nas unidades remotas a partir de políticas e regras demandadas do controlador central; ii) realizar uma análise profunda do tráfego de dados para identificar *malwares* e ações de hackers e iii) configurar remotamente os equipamentos e serviços para adequação as técnicas de transição do protocolo IPv4 para o IPv6.

O processo de implementação do encadeamento dos serviços através da distribuição ODL-SFC se inicia com a instanciação do controlador e da interface web de gerenciamento e o próximo passo consiste na inicialização do agente SFC, uma aplicação desenvolvida em Phyton que tem a funcionalidade de receber requisições do controlador e criar, dentro do plano de dados, às Funções de Serviço (SF) e os Encaminhadores de Funções de Serviço (SFF), bem como estabelecer uma relação entre os Caminhos de Funções de Serviços (SFP) e os Caminhos de

Serviços Renderizados (RSP). As informações para criação dos componentes são enviados para o controlador ODL através da solução JSON que transportar os campos de configurações com seus respectivos dados até a próxima camada. A Figura 11 ilustra as cadeias de serviços criadas a partir das funções de serviços existentes.

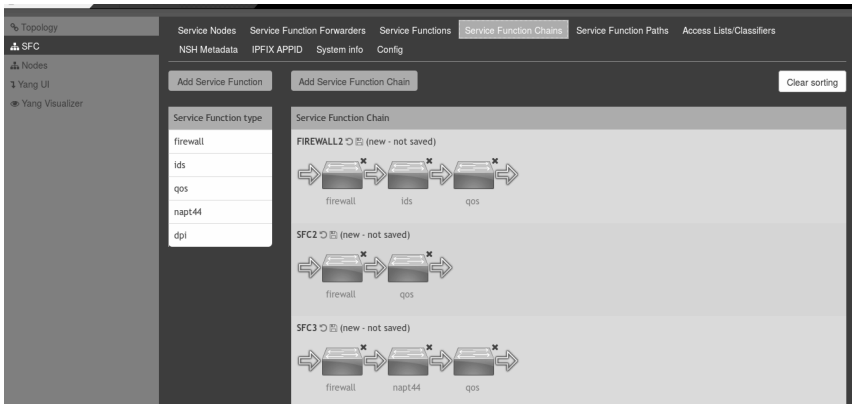


Figura 11: Interface de gerenciamento da Cadeia de Serviço, autoria própria

Depois de criadas às SF e os SFF é necessário criar uma relação ordenada de encadeamento entre as SF chamadas de cadeias de serviços, que poderão ser associadas a um plano de serviço em um SFF para efetivamente estabelecer o caminho de encadeamento entre as funções de rede e o fluxo de dados. A verificação do processo de encadeamento dos serviços na rede foi simulada com a inicialização de uma aplicação cliente no contexto do emulador Mininet e a entrada do comando (3). O fluxo produzido pelos pacotes passando pela cadeia de serviços é registrado nos logs do agente SFC, como pode ser visualizado na Figura 12.

$$> sff\_client.py\ --remote-sff-ip\ 10.0.1.4\ --remote-sff-port\ 4789\ --sfp-id\ 1 \qquad (3)$$

$INFO$:$common.services$:$SFFServer$:$Receivedapacketfrom$:('10.0.1.4',5000)
$INFO$:$common.services$:$SFFServer$:$Processingpacketfrom$:('10.0.1.4',5000)
$INFO$:$common.services$:$SFFServer$:$Sendingpacketsto$:('10.0.1.4',40001)
$INFO$:$common.services$:$firewallservicereceivedpacketfrom$('10.0.1.4',4789):
$INFO$:$common.services$:$firewall$:$Processingreceivedpacket$
$INFO$:$common.services$:$firewall$:$Sendingpacketsto$('10.0.1.4',4789)
$INFO$:$common.services$:$SFFServer$:$ListeningforNSHpackets...$
$INFO$:$common.services$:$idsservicereceivedpacketfrom$('10.0.1.4',4789):
$INFO$:$common.services$:ids:$Processingreceivedpacket$
$INFO$:$common.services$:ids:$Sendingpacketsto$('10.0.1.4',4789)
...
$INFO$:$common.services$:$SFFServer$:$Finishedprocessingpacketfrom$:('10.0.1.4',5000)
$INFO$:$common.services$:$SFFServer$:$Endofpath$

Figura 12: Captura do fluxo de encadeamento no agente SFC, autoria própria

5 CONCLUSÃO

Frente as soluções tradicionais para disponibilização de serviços nas redes de telecomunicações, os resultados obtidos durante os experimentos de integração das soluções SDN e NFV com o controlador ODL-SFC, demonstram uma maior abstração no gerenciamento e uma diminuição da complexidade de configuração dos dispositivos com ganhos significativos no tempo de disponibilização dos serviços. Essa agilidade melhora o gerenciamento em dois aspectos: primeiro, cadeias de serviços podem ser facilmente criadas a partir de funções de serviços existentes e em segundo lugar, os recursos de redes usados para entregar uma SF podem ser alocados

dinamicamente durante sua execução. Neste sentido, o emprego dessa nova abordagem possibilita uma forma centralizada e dinâmica para minimizar os custos de operação e otimizar o provimento de serviços em ambientes de *data centers*, *cloud computing* e *Internet Service Provider (ISP)*.

Como trabalho futuro é proposta a realização de estudos para aplicação da solução NSC nos dispositivos na ponta da linha das redes, conhecidos como equipamentos dentro das instalações do cliente, do inglês *Customer Provided Equipment (CPE)*, que atualmente também realizam funções de rede (e.g., roteamento, firewall, DHCP, NAT e comutação) semelhantes aos ambientes citados acima. Esta nova abordagem, objetiva adicionar ou atualizar as função de rede nos CPE's remotos, principalmente no período atual de transição dos protocolos IPv4 para o IPv6, onde existe a necessidade em aplicar técnicas de transição (e.g., pilha dupla, túneis e tradução) para manter o acesso aos serviços nos dois protocolos simultaneamente.

Agradecimentos

Aos professores da especialização em redes de computadores da Faculdade Santo Agostinho (FSA), Alberto Viegas, Amélia Acácia de Miranda Batista, Humberto Caetano Cardoso da Silva, Márcio de Melo Souza, Paulo Perris e Ricardo Almeida, por toda dedicação e motivação na transmissão do conhecimento para a formação profissional de seus alunos. E aos meus orientadores, Ricardo Gomes Queiroz e Christian Esteve Rothenberg, pela paciência, confiança e empenho que aplicaram na realização deste trabalho.

Referências

[1] SIMõES, G. *A transformação da cloud e as inovações trazidas pela IoT mudam o cenário do setor de telecomunicações*. [S.l.], 2016.

[2] COSTA, L. R. Openflow e o paradigma de redes definidas por software. *Monografia de Graduação ? Instituto de Ciências Exatas ? Departamento de Ciência da Computação - Universidade de Brasília.*, Universidade de Brasília, May 2013.

[3] MCKEOWN, N. et al. Openflow: Enabling innovation in campus networks. 2008.

[4] JOHN, W. et al. Research directions in network service chaining. 2013.

[5] FARIAS, F. N. N. et al. Pesquisa experimental para a internet do futuro: Uma proposta utilizando virtualização e o framework openflow. *XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC 2011*, SBRC 2011, v. 35, n. 15, p. 7, 2011.

[6] ZHANG, H. Clean slate design approach to networking research. May 2005.

[7] CARPENTER, B. *Middleboxes: Taxonomy and Issues*. [S.l.], 2002.

[8] FEAMSTER, N.; REXFORD, J.; ZEGURA, E. The road to sdn: An intellectual history of programmable networks. *Queue*, v. 11, n. 12, p. 20:20?20:40, Dec 2013.

[9] WATSON, G.; MCKEOWN, N.; CASADO, M. The netfpga project. 2007.

[10] ASTUTO, B. N. et al. A survey of software-defined networking: Past, present, and future of programmable networks. 2013.

[11] CASADO, M. et al. Sane: A protection architecture for enterprise networks. 2006.

[12] GORANSSON, P. *Software Defined Networks A Comprehensive Approach*. 1th. ed. [S.l.]: Elsevier, 2014.

[13] BOUCADAIR, M.; JACQUENET, C. *Software-Defined Networking: A Perspective from within a Service Provider Environment*. [S.l.], 2014, 1-56 p.

[14] NADEU, T. D.; GREY, K. *SDN: Software Defined Networking*. 1th. ed. [S.l.]: O'Reilly, 2013.

[15] SANCHEZ, A.; SZARKOWICZ, K. G. *MPLS in the SDN Era*. 1th. ed. [S.l.]: O'Reilly, 2015.

[16] LI, Y.; CHEN, M. Software-defined network function virtualization: A survey. *IEEE Access*, September 2015.

[17] BROWN, G. Serive chaining in carrier networks. 2015.

[18] RUCKERT, J. et al. Demo: Software-defined network service chaining. 2015.

[19] HALPERN, J.; PIGNATARO, C. *Service Function Chaining (SFC) Architecture*. [S.l.], 2015.

[20] QUINNG, P.; ELZUR, U. *Network Service Header (NSH) Context Header Allocation (Network Security)*. [S.l.], 2016.

Sistema automático de irrigação de baixo custo

Rodrigo Teixeira de Melo[1]
José Vigno Moura Sousa[1]
Aratã Andrade Saraiva[2]
Antonio de Macedo-Filho[1]

Resumo: O sistema automático de irrigação administra de forma eficiente os recursos hídricos da plantação para economizar água e aperfeiçoar o cultivo. A alta demanda destes recursos na agricultura irrigada traz a necessidade de economizar água nas plantações, sendo os produtores familiares os mais afetados, com menos recurso para investir em tecnologias que garantam um controle eficiente da rega. Um sistema automático de baixo custo, portanto, possibilita o acesso de destes produtores a estas tecnologias. A irrigação, quando bem realizada, é uma garantia de um bom cultivo em regiões com baixa periodicidade ou falta de regularidade das chuvas. Este sistema é desenvolvido em plataforma open source Arduino, com sensores eletrônicos que tem a capacidade de realizar leituras de temperatura ambiente, umidade do solo e umidade do ar, e com base nestas informações executar ações de irrigação.

***Palavras-chave**:* Agricultura familiar. Baixo custo. Sistema de irrigação.

Abstract: *The automatic irrigation system efficiently manages water resources of the plantation to save water and improve cultivation. The high demand of these resources in irrigated agriculture brings the need to save water in plantations, being family farmers most affected, with less resource to invest in technologies that ensure efficient irrigation control. A low cost automatic system, therefore, will provide access of these producers to these technologies. Irrigation, when done well, is a guarantee of a good crop in regions with low frequency or lack of regular rainfall. This system is developed in open source Arduino platform with electronic sensors that have the ability to perform temperature, soil moisture and humidity readings, and based on this information, execute irrigation actions.*

***Keywords:** Familiar farming. Irrigation System. Low cost.*

1 Introdução

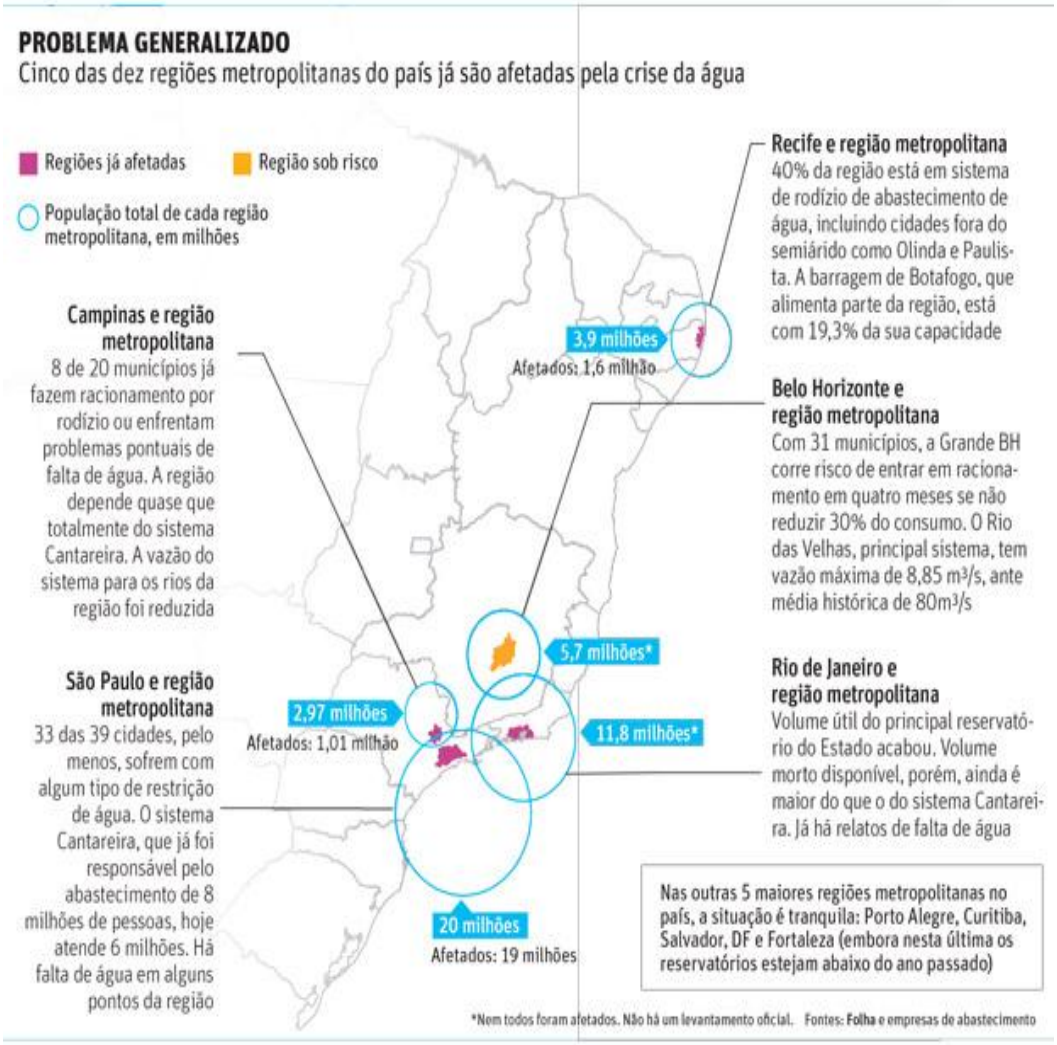
A cada dia que passa os recursos hídricos disponíveis estão ficando de difícil acesso e aumentando o custo relativo. Estes recursos são utilizados de muitas maneiras, sendo em uma grande porcentagem na agricultura [1]. Com o custo desse bem tão vital para a vida aumentando é indispensável o desenvolvimento de técnicas e tecnologias para a redução do desperdício e aumento da produtividade hídrica na irrigação.

Segundo o Censo Agropecuário de 2006 do IBGE [2] "Apesar de cultivar uma área menor com lavouras e pastagens (17,7 e 36,4 milhões de hectares, respectivamente), a agricultura familiar é responsável por garantir boa parte da segurança alimentar do país, como importante fornecedora de alimentos para o mercado interno", ainda assim, a agricultura familiar não detém de tanta tecnologia no processo do cultivo. Predominantemente, os agricultores ainda têm pouco acesso às técnicas necessárias à produção sustentável [3].

[1] Bacharelado em Ciência da Computação, Universidade Estadual do Piauí, UESPI - Campus Piripiri, Av. Pres. Castelo Branco, 180, Piripiri – PI, 64260-000, Brasil.
{morph30s@hotmail.com, josevigno@gmail.com, amfilho@gmail.com}
[2] GALILEO – CEUMA, Teresina – PI, Brasil.
{aratasaraiva@gmail.com}

Somado ao pouco acesso às tecnologias de gestão de recursos na agricultura familiar, existe ainda outro fator crítico para a produção em pequena (e também em larga) escala, a seca. Em locais em que as chuvas não têm quantidade e nem periodicidade satisfatórias, como por exemplo, na região nordeste, é comum, e muitas vezes extremamente necessário o uso da irrigação no cultivo, porém, é essencial que os recursos hídricos sejam geridos da melhor forma possível para se evitar desperdícios e desgaste do solo. Isso porque a qualidade da água aplicada durante a estação de cultivo, bem como a quantidade, não só influencia o rendimento das culturas, como também modifica as propriedades físicas, químicas e microbiológicas dos solos; e isto pode afetar sensivelmente a produtividade obtida nos anos subsequentes [4]. Na Figura 1 são apresentados os números de pessoas e municípios afetados pela seca nas principais regiões metropolitanas do Brasil em 2015 [5].

Figura 1: Números da seca nas principais regiões metropolitanas em 2015.



Um bom sistema de irrigação é, portanto, uma garantia de uma gestão eficiente dos recursos hídricos durante as diferentes fases de maturação da cultura, mas no geral, o maquinário agrícola moderno, além de caro, pode ser de uso complexo e de difícil acesso, fatores estes, que acabam por afastar o produtor familiar do que há de novo em tecnologia no setor agrícola.

Podemos destacar como um bom exemplo de sistema automático de irrigação, o projeto *iRain* [6], que visa o controle eficiente de jardins urbanos por meio de uma rede de sensores e atuadores utilizando a tecnologia *Zigbee* (padrão para transmissões sem fio de baixa velocidade, baixa potência e baixo custo em arquiteturas *mesh*, definido pelo padrão IEEE 802.15.4). Sendo este, capaz de realizar leituras de previsão do tempo e prevenir a irrigação em caso de precipitação prevista para até 03 dias, além de manter os níveis de umidade do

solo por meio de sensores e dar preferência para a irrigação pela noite e períodos de vento fraco.

Ainda utilizando a tecnologia *Zigbee* e o conceito de redes de sensores sem fio (WSN), um sistema foi desenvolvido com o intuito de gerenciar de forma inteligente uma plantação na China, como sendo uma aplicação que oferece uma metodologia eficiente para o monitoramento remoto em sistemas inteligentes de irrigação de larga escala [7].

Apesar de ser uma tecnologia eficiente e de baixo custo, os módulos compatíveis com a tecnologia *Zigbee*, como o módulo *Xbee*, ainda têm seu preço relativamente alto no Brasil, alguns chegando a custar R$ 250,00 a unidade, inviabilizando seu uso, por este fugir do escopo do orçamento deste trabalho. Diante disso, este estudo tem por objetivo desenvolver um sistema de irrigação flexível, de baixo custo e de fácil utilização, capaz de manter o nível adequado de umidade do solo tomando como parâmetro as condições de temperatura e umidade do solo considerado ideal para o período e tipo de cultura, que são fornecidas pelo usuário. A meta orçamentária é de R$ 100,00 e o protótipo inicial traz, também, a possibilidade de se trabalhar com horários predeterminados para irrigação, onde o usuário pode definir por meio de uma interface física, a melhor hora para se efetuar a irrigação durante o dia.

2 Lista de materiais

A execução deste trabalho é baseada na plataforma Arduino [8], que é uma plataforma de hardware e software open source, cuja principal vantagem é a facilidade em criar projetos interativos nas áreas da robótica, automação, monitoramento, etc. devido à grande quantidade de material disponível para estudo e simulação, e sua linguagem de fácil compreensão e alto nível de abstração (originalmente baseada na linguagem C/C++), além de diversos componentes eletrônicos compatíveis. Os materiais utilizados podem ser divididos em módulos, agrupados conforme o seu papel durante a prototipagem (Tabela 1).

Tabela 1: Módulos do protótipo

Nome	Componentes	Descrição
Módulo de controle	Placa de desenvolvimento Arduino UNO.	O módulo principal, responsável por controlar e intercomunicar com os demais módulos.
Módulo de sensoriamento	Sensores de temperatura ambiente, umidade do ar, e umidade do solo.	Realiza a captação de variações no ambiente e converte-as em sinais elétricos para processamento posterior.
Módulo de interface	Visor LCD, botões e LEDs indicadores.	Permite monitorar e definir parâmetros para a irrigação.
Módulo de atuação	Válvulas solenoides, relés, transistores de potência, contatores, etc.	Está ligado diretamente ao dispositivo dispersor de água.

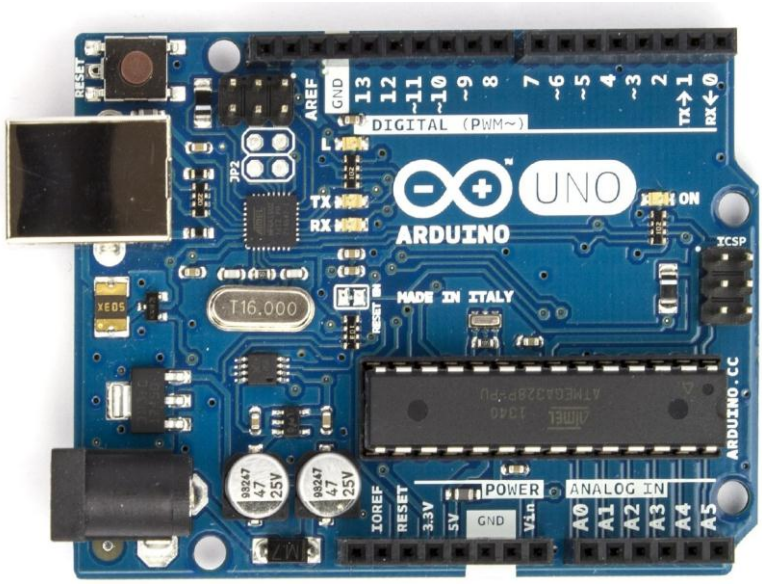
2.1 Módulo de controle

O módulo de controle é composto essencialmente pela placa de desenvolvimento Arduino UNO R3, que utiliza o micro controlador ATmega328P de 8 bits, possui 14 entradas/saídas digitais, das quais 6 podem ser utilizadas como saídas PWM (Pulse Width Modulation), 6 entradas analógicas, conexão USB, 32KB de memória flash e memória SRAM (Static Random Access Memory) de 2KB, além de uma memória EEPROM de 1KB, extremamente útil para armazenamento de determinados parâmetros de configuração, no caso deste trabalho. O micro controlador opera numa frequência de 16MHz.

A escolha desta placa de desenvolvimento se deve em suma pelo seu baixíssimo valor de aquisição (entre R$ 20,00 e R$ 70,00, dependendo do fornecedor), e também por sua popularidade e facilidade de manutenção, visto que o micro controlador (chip retangular na parte inferior direita, na Figura 2) pode ser substituído por

outro apenas por meio de encaixe. Além disso, é capaz de prover processamento mais que suficiente para as demandas deste protótipo.

Figura 2: Arduino UNO R3.



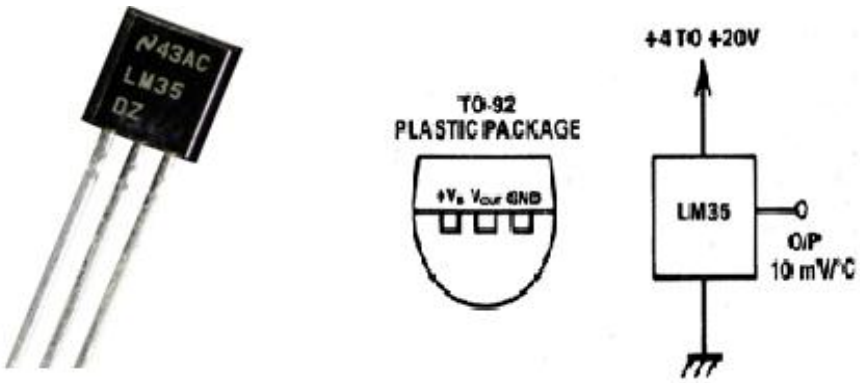
2.2 Módulo de sensoriamento

Este módulo conta com três sensores, responsáveis por monitorar a temperatura ambiente, umidade do ar, e umidade do solo.

2.2.1 Sensor de temperatura LM35

O sensor LM35 monitora a temperatura ambiente, além de possuir uma boa precisão, tem custo irrisório (aproximadamente R$ 3,00 à R$ 7,00) e é facilmente encontrado em lojas de componentes eletrônicos (Figura 3). É de fácil utilização, pois ao ler seu terminal de saída, cada 10mV corresponde à 1ºC, desse modo, ao medir no terminal de saída do sensor uma tensão de 345mV, significa dizer que o ambiente em que está situado enfrenta a temperatura de 34.5ºC, podendo assim ser fazer medições utilizando apenas um voltímetro. Seu uso no protótipo se dá como uma alternativa de maior precisão e de menor custo financeiro para leituras de temperatura, já que sua margem de erro é consideravelmente menor (± 0,5ºC, contra ± 2ºC do DHT11).

Figura 3: LM35 (à esquerda) e seu diagrama de pinos (no centro) e representação eletrônica (à direita).



2.2.2 Sensor de umidade e temperatura DHT11

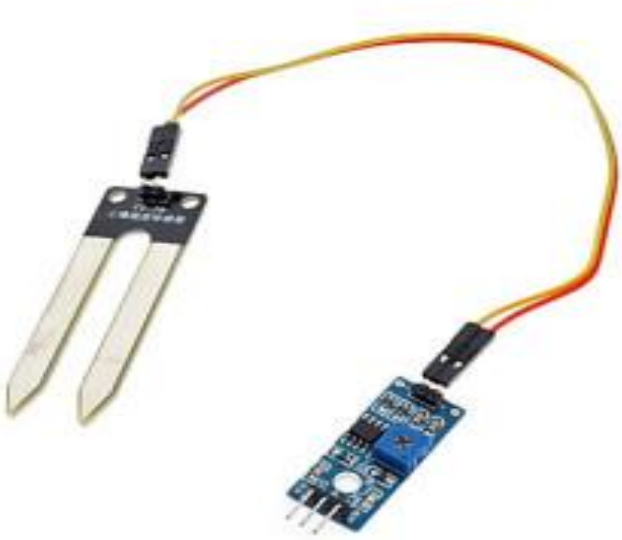
O sensor digital é capaz de medir com relativa precisão a temperatura ambiente e umidade do ar. O custo do sensor é em torno de R$ 20,00, este sensor pode fazer medições de umidade relativa do ar entre 20% a 90% com precisão de ± 5%, e leituras de temperatura ambiente entre 0ºC a 50ºC, com precisão de ± 2ºC, já possui biblioteca pronta para a plataforma Arduino, podendo ser facilmente utilizado (Figura 4).

Figura 4: Módulo sensor DHT11.

2.2.3 Sensor de umidade do solo

O sensor de umidade do solo é composto por duas hastes de prova, que vão diretamente no solo, têm seu princípio de funcionamento baseado na resistividade elétrica do solo, ou seja, à medida que o solo é umedecido, a sua resistência elétrica diminui, permitindo a corrente fluir da haste energizada, para a haste de medição, conectada ao controlador principal (Figura 5). Tem custo médio de R$ 20,00, porém pode ser feito utilizando apenas duas hastes metálicas, galvanizadas, isoladas uma da outra, para reduzir ainda mais seu custo. As hastes deste modelo tem comprimento de 6cm e 2cm de espessura.

Figura 5: Sensor de umidade do solo.



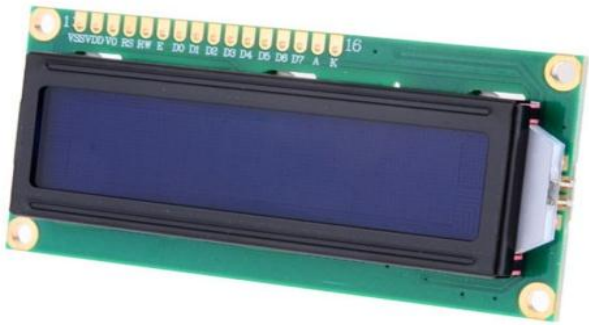
2.3 Módulo de interface

O módulo é responsável por trazer ao usuário flexibilidade em ajustar parâmetros para a irrigação, permitindo que o sistema se adapte à diversas situações sem a necessidade de reprogramação. Este conta com botões para ajustes, e um visor LCD para monitoramento em tempo real.

2.3.1 Visor LCD

Na Figura 6 é mostrado o visor hitachi HD44780 de 16x2 caracteres, possui uma biblioteca padrão no ambiente de desenvolvimento Arduino e permite escrever até 32 caracteres simultaneamente, possibilitando mostrar em tempo real os dados coletados pelos sensores, bem como o status do sistema em geral.

Figura 6: Visor de matriz de pontos LCD.



2.4 Módulo de atuação

O módulo de atuação responde pela etapa de potência do protótipo, pode ser constituída de relés (Figura 7.a), contatores (Figura 7.b), válvulas solenoides (Figura 7.c), transistores de potência, etc. dependendo do que se for utilizar para efetuar o deslocamento de água sobre o campo a ser irrigado. No caso de um sistema de irrigação por aspersão, que geralmente se utilizam bombas de água para prover a pressão necessária para produzir jatos, o indicado seria utilizar chaves eletromecânicas, como relés ou contatores, dependendo da demanda energética da moto bomba.

Em testes em laboratório, este módulo é substituído por LED para evitar acidentes com eletricidade, devido às altas tensões de operação destes componentes. Logo, no protótipo utiliza-se apenas um LED para simular a ativação da etapa de potência.

Figura 7: a) Módulo Relé; b) Contator; c) Válvula solenoide.

3 Hardware e implementação do software

Esta seção consiste em integrar os todos os módulos citados e realizar testes de hardware e software para verificar a integridade e precisão dos componentes quando trabalhando em conjunto, já que a placa Arduino UNO fornece no máximo 40mA em cada terminal e o conjunto inteiro não pode fornecer mais que 200mA. Sendo assim necessário dimensionar corretamente os dispositivos, para que não haja concorrência energética, o que pode acarretar em mau funcionamento, imprecisão, ou mesmo danos permanentes aos dispositivos integrados.

3.1 Estabilização do sensor LM35

O sensor LM35 demonstra diversas oscilações quando operando em conjunto com os demais componentes. Isso se deve em suma, a sua alta sensibilidade a variações de corrente, necessitando assim de uma rotina de amortização em código, que consiste em considerar como informação válida de temperatura, apenas a média aritmética de diversas leituras. No caso deste protótipo, os resultados se mostraram satisfatórios ao realizar uma média entre as 10 primeiras leituras do sensor, a rotina é exemplificada em pseudocódigo adiante.

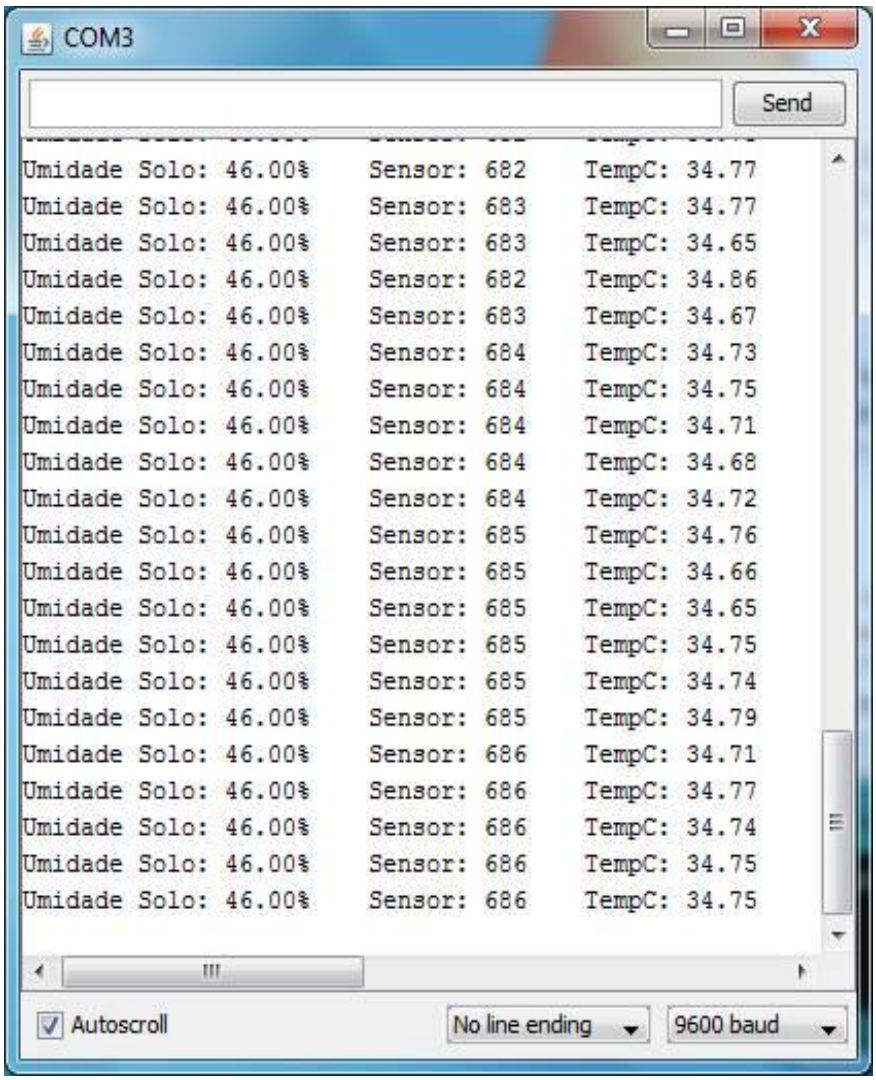
```
1. var temperatura, mediaLeitura, i, valorSensor;
2. begin;
3. for (i = 1; i<= 10; i++){
4.    valorSensor = sensorRead();
5.    mediaLeitura = mediaLeitura + valorSensor;
6. }
7. mediaLeitura = mediaLeitura / 10;
8. temperatura = mediaLeitura * 0.488;
9. end;
```

3.2 Calibragem e mapeamento de valores do sensor de umidade do solo

O sensor de umidade do solo traz como retorno de sua leitura um valor entre zero (extremamente úmido) e 1023 (extremamente seco), esta faixa de valores corresponde à capacidade do conversor analógico-digital da placa Arduino UNO que pode ler com resolução de 10 bits. Por ser um pouco confuso avaliar a umidade do solo em valores entre zero e 1023, o programa principal realiza uma conversão desses valores em escala percentual. Sendo necessário somente calibrar o sensor conforme amostras de solo, já que o mesmo, dificilmente trará o valor zero para um solo muito úmido, pois parte da energia enviada ao sensor é dissipada. A calibragem do sensor se dá por meio de testes em diferentes amostras de solo, do mais seco ao mais encharcado, como mostra a Figura 8, os resultados obtidos são apresentados na Figura 9, onde é exibido o valor percentual, o valor obtido diretamente do sensor, e o valor de temperatura ambiente dado em graus Celsius.

Figura 8: Calibragem do sensor de umidade do solo em diversas amostras de solo.

Figura 9: Leituras retornadas pelos sensores.

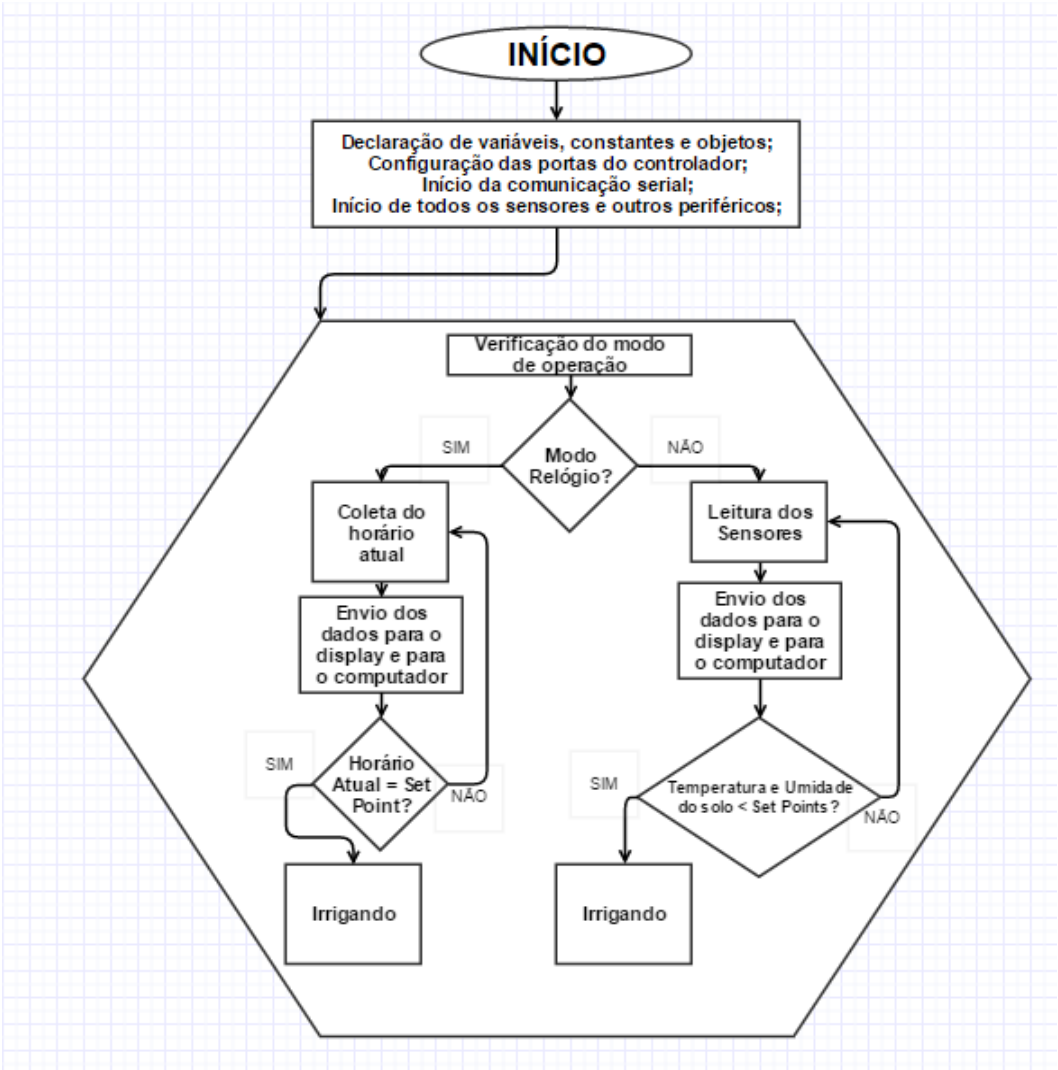


3.3 Software de controle

O sistema conta com dois modos de operação: por sensoriamento, e por horário predefinido. No modo por sensoriamento, os dados coletados pelo sensor de umidade determinam o ponto crítico da irrigação enquanto o sensor de temperatura tem função de impedir a irrigação em caso de temperaturas muito altas, que podem causar rápida evaporação da água, impedindo a absorção pela planta e causando desperdício ou até mesmo danos à cultura, e serve indiretamente como relógio, já que na região nordeste, durante o dia as temperaturas ultrapassam facilmente a casa dos 30ºC, e pela noite, tendem a ficar abaixo disso. Desta forma é possível programar o sistema para operar apenas com temperaturas mais amenas.

A execução se dá de forma sequencial, e a primeira rotina é iniciar todos os periféricos (sensores, visor, etc), configurar o comportamento das portas do controlador, em modo de entrada ou saída de dados, e iniciar a comunicação com o computador. Após isso o sistema executa uma série de instruções em loop constante (representado por um hexágono na Figura 10) até que o sistema seja desligado.

Figura 10: Fluxograma do software de controle.

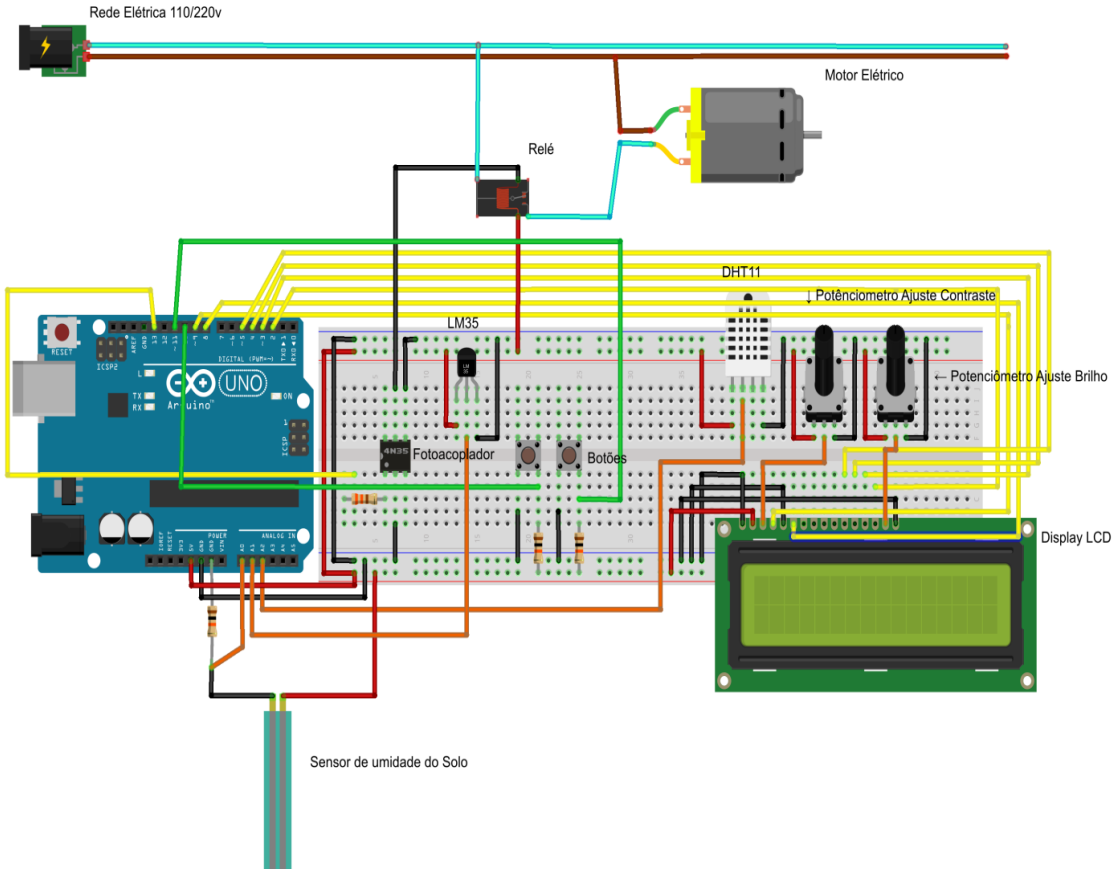


O modo relógio é baseado em uma adaptação na biblioteca do módulo RTC (*Real Time Clock*) DS1307, esta adaptação permite que o controlador grave o horário do computador que o controla, e a partir disso, use o próprio oscilador interno para contar as horas, minutos e segundos. O módulo RTC não foi utilizado de fato no protótipo, em decorrência de o único disponível para a execução deste trabalho, ter sido adquirido já avariado de fábrica, portanto optou-se por simular o relógio no próprio controlador.

Como exibido na Figura 10 o programa avalia constantemente dois parâmetros chamados set points, estes parâmetros, são valores críticos de umidade do solo e temperatura, que são definidos pelo usuário. O software verifica a cada iteração, se a umidade do solo caiu abaixo do valor predefinido pelo usuário, e caso a temperatura também esteja abaixo do limite, o software então envia um sinal de saída para o módulo de atuação, iniciando o processo de irrigação. Depois de um intervalo de tempo, o software verificará novamente se o nível de umidade do solo se estabeleceu acima do set point, caso a condição seja verdadeira, o controlador desligará o módulo de atuação e aguardará o próximo momento. A etapa de irrigação só é iniciada se e somente se as duas condições propostas forem verdadeiras, mesmo que o solo esteja com os níveis de umidade abaixo do que o usuário definiu como ideal, o sistema não iniciará a irrigação, caso a temperatura esteja acima do limite estabelecido.

Já no modo relógio, o software fica constantemente verificando se o horário atual coincide com o horário que o usuário definiu para efetuar a irrigação, se sim, a irrigação é iniciada por um intervalo de tempo, e quando encerrada, o sistema aguardará o próximo dia para efetuar novamente a irrigação.

Figura 11: Diagrama do protótipo, desenhado no software Fritzing.



A Figura 11 retrata as conexões entre os componentes utilizando no módulo de atuação um relé, que atua como chave de um motor elétrico ligado na rede de maior potência. No centro da imagem, estão os botões para ajustes. Pressionar e segurar o botão do lado esquerdo, permitirá ajustar o set point para o sensor de umidade do solo, os valores serão incrementados em uma faixa de 15% a 70%, e ao soltar o botão, o valor é gravado na memória e o programa retoma a execução considerando o novo parâmetro fornecido. O mesmo procedimento vale para o botão da direita, mas este é responsável por ajustar o set point de temperatura, e possui uma faixa de valores possíveis entre 0ºC e 38ºC.

Ao pressionar os dois botões simultaneamente, o programa irá alterar o sistema entre o modo de sensoriamento e o modo relógio, o ultimo modo mostrado no display, é assumido como modo padrão ao soltar os botões. No modo relógio, os botões esquerdo e direito terão função de ajustar a hora e o minuto, respectivamente, em que o sistema irá realizar a aplicação da lâmina d'água.

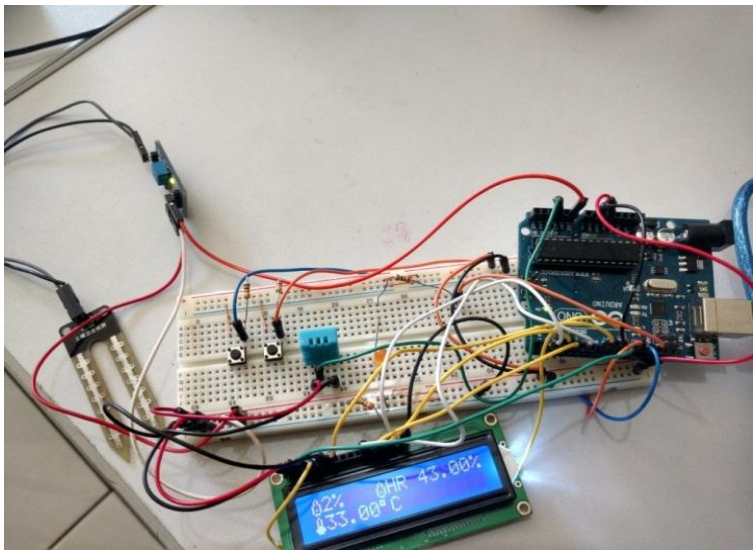
3.4 Resultados

O protótipo é montado em uma de matriz de contatos (Figura 12) para facilitar a etapa de testes, deste modo, é possível apenas encaixar os componentes na matriz e verificar se o sistema funciona como deveria. Ainda na Figura 12, é possível perceber que o sistema está operante, conforme o esperado em sua tela inicial, mostrando no início da primeira linha, os dados obtidos pelo sensor de umidade do solo, em 2%, revelando que o sensor precisa de recalibragem. Logo mais à direita, na primeira linha, o display mostra o nível de umidade relativa do ar (HR) em 43%, e logo abaixo, a temperatura ambiente em graus Celsius.

Os testes são realizados com componentes de baixa potência, a fim de se evitar possíveis acidentes envolvendo descargas elétricas, já que o controlador principal funciona com 5V (corrente contínua) e fornece no máximo 0,2 Watts em cada terminal, e uma bomba d'água convencional (que geralmente trabalha na rede mono ou até trifásica) exige bem mais potência para funcionar, e exige também maior cuidado em sua manipulação

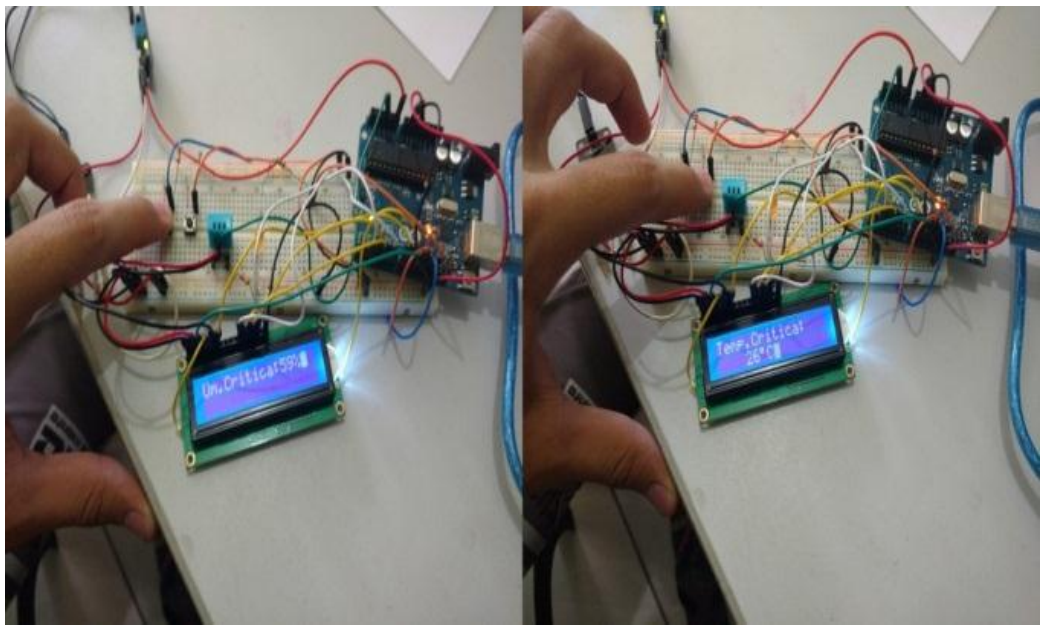
devido a alta tensão de operação. Entretanto, o protótipo demonstra total capacidade de atuar em circuitos de alta potência, adequando-se assim aos mais diversos tipos de irrigação.

Figura 12: protótipo montado em placa de matriz de contatos.



Na Figura 13, verifica-se que os botões de ajuste funcionam corretamente, possibilitando alterar os set points, conforme programado.

Figura 13: Teste dos botões para ajuste dos set points.



Como verificado na Figura 14, a interface de ajuste do sistema opera de modo que se possa definir a melhor maneira de o conjunto atuar no ambiente em que se situa, podendo alterar o seu modo de operação apertando os dois botões simultaneamente. Na Figura 15 é simulado um ambiente onde as condições seriam satisfatórias para iniciar a irrigação, e como era esperado, o sistema reagiu mandando um pulso elétrico ao LED laranja no centro da matriz de contatos. Este mesmo pulso elétrico, é suficiente para ativar qualquer circuito de potência, como uma moto bomba para sistemas de irrigação.

Figura 14: Alterando o sistema para o modo relógio.

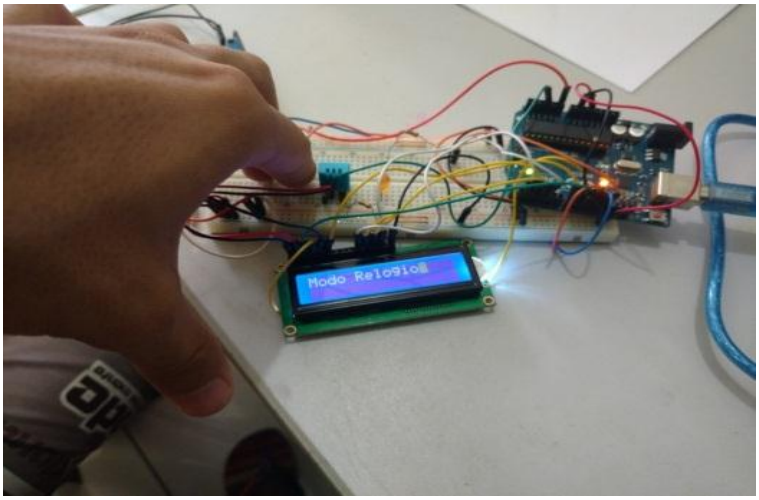
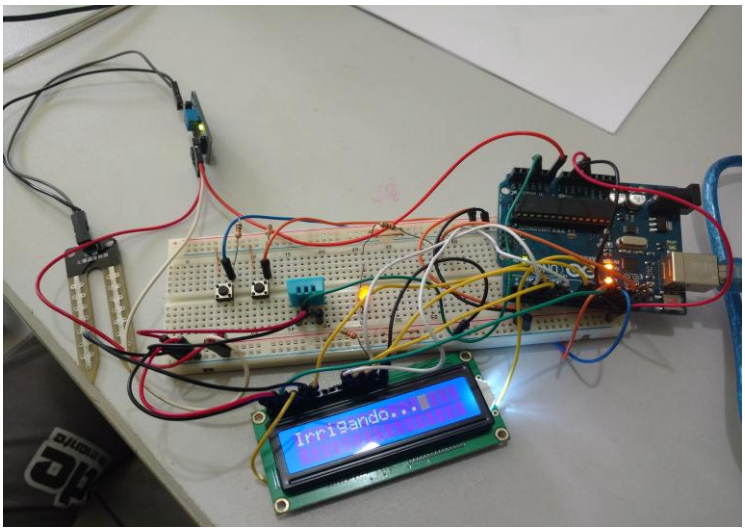


Figura 15: Teste simulado de ativação do sistema.



Da mesma forma, o sistema executa a o procedimento de irrigação no modo relógio, enviando o pulso elétrico ao LED sempre que a hora atual coincide com a hora definida para a irrigação, trazendo flexibilidade ao sistema, permitindo que este opere em horários predefinidos, podendo assim efetuar a irrigação automaticamente até em períodos noturnos, em que não é comum ter pessoas disponíveis para fazê-la manualmente.

4 Custo do protótipo

É exibido na Tabela 2 o custo com hardware do protótipo, o módulo de atuação fica fora do orçamento, pois este varia conforme o equipamento que se for utilizar. Uma motobomba com potência elétrica de 1.000 Watts por exemplo, poderia ser acionada com um simples relé, que custa em média R$ 5,00 (desconsiderando circuitos de proteção, etc). Por outro lado, uma bomba de potência superior à 10.000 Watts necessitaria de um circuito de ativação mais elaborado e de maior valor financeiro. Demais custos como fiação, circuitos de proteção, caixas herméticas, etc. também ficam de fora do orçamento. Esta seção abrange portanto apenas os componentes do conjunto de controle.

Tabela 2: Custo total do protótipo

Nome	Quantidade	Valor unitário	Valor total
Arduino UNO R3	1	R$ 54,90	R$ 54,90
Sensor DHT11	1	R$ 12,90	R$ 12,90
Sensor LM35	1	R$ 6,05	R$ 6,05
Sensor de Umidade do solo	1	R$ 16,59	R$ 16,59
Visor LCD	1	R$ 11,99	R$ 11,99
Botão tipo pushbutton	2	R$ 0,10	R$ 0,20
LED para teste	1	R$ 0,10	R$ 0,10
Total			**R$ 102,73**

Conforme demonstrado na Tabela 2, o custo do protótipo excedeu a meta orçamentária em R$ 2,73, entretanto, é importante ressaltar que foram utilizados componentes originais, vendido em sites de eletrônicos como solução pronta, como a placa de prototipagem Arduino UNO. Estes componentes podem ter seu custo reduzido em relação aos que são vendidos nas lojas. Quanto a placa Arduino UNO, é possível montá-la de forma *stand-alone*, reduzindo seu tamanho e a quantidade de componentes, e reduzindo também seu preço. A abordagem *stand-alone* limita o conjunto ao próprio controlador ATmega328P e alguns poucos componentes, reduzindo seu preço a um valor médio de R$ 20,00. Enquanto ao sensor de umidade do solo, pode-se reutilizar hastes galvanidazas e um resistor de 10kΩ, reduzindo seu custo a menos de R$ 5,00. Com esta abordagem, estima-se que o protótipo teria seu custo em torno de R$ 57,00.

5 Considerações finais

Diante do exposto, o protótipo de sistema de irrigação de baixo custo atingiu os seus objetivos realizando de forma automática a irrigação nos horários definidos e de acordo com a necessidade avaliada na medição da temperatura, umidade do ar e do solo em laboratório, possibilitando ainda o ajuste dos parâmetros de irrigação por meio de interface física, o que possibilita que o sistema se adeque às diferentes fases de maturação da cultura sem a necessidade de reprogramação. Além disso, o sistema é acessível aos produtores familiares, por ser desenvolvido em plataforma open source, utilizando componentes de baixo custo.

Como trabalhos futuros será expandido os testes para uma plantação rural familiar e desenvolvido a interface para o usuário final. Na versão final espera-se reduzir o tamanho do protótipo com uma placa única de controle, já que o controlador ATmega328P da placa Arduino UNO, pode operar perfeitamente em modo *stand-alone*, reduzindo o circuito controlador ao tamanho aproximado do próprio chip, requerendo apenas alguns poucos componentes.

Agradecimentos

Aos professores e companheiros de pesquisa da Universidade Estadual do Piauí e do Instituto Galileo por contribuírem com ideias e equipamentos para a realização deste trabalho.

Referências

[1] EBC. Agricultura é quem mais gasta água no brasil e no mundo, 2013. Disponivel em: <http://www.ebc.com.br/noticias/internacional/2013/03/agricultura-e-quem-mais-gasta-agua-no-brasil-e-no-mundo>. Acesso em: 16 jun. 2016.

[2] IBGE. Censo Agropecuário 2006, Rio de Janeiro, 2006. Disponivel em: <http://biblioteca.ibge.gov.br/visualizacao/periodicos/50/agro_2006_agricultura_familiar.pdf>. Acesso em: 16 jun. 2016.

[3] FEIDEN, A. Tecnologias para a agricultura familiar, 2014. Disponivel em: <http://ainfo.cnptia.embrapa.br/digital/bitstream/item/103482/1/DOC2014122.pdf>. Acesso em: 16 jun. 2016.

[4] AMORIM, J. R. A. D. Salinidade em áreas irrigadas: origem do problema, conseqüências e possíveis soluções. Disponivel em: <http://www.grupocultivar.com.br/artigos/salinidade-em-areas-irrigadas-origem-do-problema-consequencias-e-possiveis-solucoes>. Acesso em: 16 jun. 2016.

[5] UOL. Seca já atinge 5 das 10 maiores regiões metropolitanas do país, 2015. Disponivel em: <http://www1.folha.uol.com.br/cotidiano/2015/01/1580032-seca-ja-atinge-5-das-10-maiores-regioes-metropolitanas-do-pais.shtml>. Acesso em: 30 jun. 2016.

[6] CAETANO, F.; PITARMA, R.; REIS, P. Intelligent management of urban garden irrigation, 2014. ISSN 978-9-8998-4343-1.

[7] ZHOU, Y. et al. A Wireless Design of Low-Cost Irrigation System Using ZigBee Technology, 2009. ISSN 978-0-7695-3610-1.

[8] ARDUINO. What is Arduino?, 2016. Disponivel em: <https://www.arduino.cc/en/Guide/Introduction>. Acesso em: 16 jun. 2016.

Uma abordagem sobre engenharia de requisitos para metodologia ágil SCRUM

Adriana F. da Silva [1]
Francisca Raquel de V. Silveira [1]

Resumo: A engenharia de requisitos fornece o mecanismo para entender o que o cliente deseja, realizando a elicitação, análise e gerência de requisitos, porém ainda pode surgir problema nessa fase. Para tentar tratar esses problemas temos as metodologias ágeis que lidam com requisitos a fim de implementá-los corretamente e satisfazer as necessidades do cliente. Dessa forma, existem práticas que auxiliam na comunicação com o cliente e equipe de desenvolvimento, melhorando a elicitação dos requisitos, entrega do produto no tempo adequado, reduzindo custo, priorizando a qualidade e funcionalidades. Este trabalho apresenta um estudo sobre a engenharia de requisitos para a metodologia ágil Scrum, apresentando as práticas e as restrições do Scrum e as técnicas tradicionais da engenharia de requisitos. Além de relatar um estudo de caso aplicado em um projeto que utiliza metodologia ágil, repassando uma visão de como a relação de métodos ágil e engenharia de requisitos está se apresentando no desenvolvimento de software.

***Palavras-chave*:** Engenharia de Requisitos. Metodologia Ágil. Scrum.

Abstract: *The requirements engineering provides the mechanism to understand the customer desire, making the collection, analysis, and manage of requirements, but can still arise problem in this step. To try to address these problems, we have the agile methodologies that deal with requirements in order to implement them properly and to meet customer needs. Thus, there are practices that aid in the communicating with the client and the development team, improving the collection of requirements, delivery of the product at the appropriate time, reducing cost, giving priority to quality and functionality. This paper presents a study about the requirements engineering for the agile methodology Scrum, presenting the practices and restrictions of the Scrum and the traditional techniques of requirements engineering. In addition, we reported a case study applied to a project using agile methodology, showing a vision of how the relationship of agile methods and requirements engineering is presented in the software development.*

***Keywords*:** *Engineering requirements. Agile methodology. Scrum.*

1 Introdução

A engenharia de requisitos fornece o mecanismo apropriado para entender o que o cliente deseja, analisando as necessidades, avaliando a viabilidade, negociando uma solução razoável, especificando a solução sem ambiguidade, validando a especificação e gerenciando as necessidades à medida que são transformadas em um sistema operacional [1]. O papel da engenharia de requisitos é importante para a construção e desenvolvimento de software, tendo responsabilidade de garantir qualidade, prazos, custos e funcionalidades. No entanto os mecanismos que a engenharia de requisitos fornece sempre costumam ser muito rigorosos e extensos [2].

Com o intuito de entregar produto mais rápido, com alta qualidade, satisfazer as necessidades do cliente e visando melhorar o andamento dos projetos, foram criadas as metodologias ágeis. Entre essas metodologias estão: Extreme Programming (XP), SCRUM, Feature Driven Development (FDD), Adaptive Software Development (ASD), Crystal Methods (CM) [2].

O Scrum, criado com foco no processo de gestão de projetos de software complexos, possui como principal característica a comunicação entre as equipes que participam direta ou indiretamente e não há especificações

[1]Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE) - Campus Aracati
{adricads@hotmail.com, raquel_silveira@ifce.edu.br}

referentes a práticas de construção, pois o método enfatiza que as equipes sejam auto-organizadas e os processos são orientados a objetivos e resultados [3].

Este trabalho tem como objetivo realizar um estudo de caso sobre a aplicação do processo de engenharia de requisitos em um projeto de desenvolvimento de software que utiliza metodologia ágil SCRUM. Uma pesquisa para investigar e expor técnicas e abordagens utilizadas na engenharia de requisitos para projetos com Scrum.

O presente trabalho está organizado da seguinte maneira: a seção 2 apresenta um referencial teórico sobre metodologia ágil SCRUM e engenharia de requisitos; a seção 3 expõe como é tratada a engenharia de requisitos no SCRUM; a seção 4 descreve o estudo de caso utilizado neste trabalho, detalhando a visão geral do projeto, experiências e resultados; finalmente, a seção 5 apresenta as considerações finais e os trabalhos futuros.

2 Referencial Teórico

Nesta seção apresentamos sucintamente os principais conceitos referentes a metodologia ágil Scrum, com as etapas do Sprint e seus artefatos, contextualizando ainda a engenharia de requisitos com os tipos de requisitos, os processos da engenharia de requisitos e os seus artefatos.

2.1 Metodologia Ágil: SCRUM

Em fevereiro de 2001, um grupo com 17 membros da comunidade de software se reuniram e definiram oficialmente o manifesto Ágil de Software, com o objetivo de propor melhores maneiras de desenvolver softwares. O manifesto ágil afirma valorizar "os indivíduos e interação sobre processos e ferramentas, software funcionando mais que documentação abrangente, colaboração do cliente sobre negociação de contratos, e responder as mudanças seguindo um plano"[3].

O Scrum é um processo para projetos que realizam desenvolvimento de software que seja focado nas pessoas, no qual o ambiente tenha uma mudança constante nos requisitos, podendo ser também considerado uma técnica utilizada para realizar o gerenciamento de processo para o desenvolvimento de software, além de propor entregar produtos no prazo correto, com alto grau de qualidade e visando satisfação do cliente [3].

Scrum é fundamentado nas teorias empíricas de controle de processo, ou empirismo. O empirismo afirma que o conhecimento vem da experiência e de tomada de decisões baseadas no que é conhecido. O Scrum emprega uma abordagem iterativa e incremental para aperfeiçoar a previsibilidade e o controle de riscos [4].

Três pilares apoiam a implementação de controle de processo empírico: (i) transparência, (ii) inspeção e (iii) adaptação [4].

Transparência: São aspectos significativos do processo que devem estar visíveis aos responsáveis pelos resultados. Esta transparência requer aspectos definidos por um padrão comum para que os observadores compartilharem um mesmo entendimento do que está sendo visto.

Inspeção: Os usuários Scrum devem, frequentemente, inspecionar os artefatos Scrum e o progresso em direção a detectar variações. Esta inspeção não deve, no entanto, ser tão frequente que atrapalhe a própria execução das tarefas. As inspeções são mais benéficas quando realizadas de forma diligente por inspetores especializados no trabalho a se verificar.

Adaptação: Se um inspetor determina que um ou mais aspectos de um processo desviou para fora dos limites aceitáveis, e que o produto resultado será inaceitável, o processo ou o material sendo produzido deve ser ajustado. O ajuste deve ser realizado o mais breve possível para minimizar mais desvios.

O time Scrum é composto pelo Product Owner, o time de desenvolvimento e o Scrum Master. O time Scrum é auto-organizável, escolhe qual a melhor forma para completar seu trabalho, em vez de ser dirigido por outros de fora do time. Também é multifuncional, possuem todas as competências necessárias para completar o trabalho sem depender de outros que não fazem parte da equipe [4].

O Product Owner, ou dono do produto, é o responsável por maximizar o valor do produto e do trabalho do time de desenvolvimento. O Product Owner é a única pessoa responsável por gerenciar o Backlog do Produto. Ele

tem o dever de expressar claramente os itens do Backlog do Produto, ordenar os itens do Backlog do Produto para alcançar melhor as metas e missões, garantir o valor do trabalho realizado pelo time de desenvolvimento, garantir que o Backlog do Produto seja visível, transparente, claro para todos, mostrar o que o time Scrum vai trabalhar a seguir e garantir que o time de desenvolvimento entenda os itens do Backlog do Produto [4].

O time de desenvolvimento consiste de profissionais que realizam o trabalho de entregar uma versão usável que potencialmente incrementa o produto ao final de cada Sprint. Times de desenvolvimento são estruturados e autorizados pela organização para organizar e gerenciar seu próprio trabalho. O tamanho da equipe costuma ter entre 8 a 10 integrantes, podendo variar dependendo do ambiente em que se aplica [3].

O Scrum Master é responsável por garantir que o Scrum seja entendido e aplicado. O Scrum Master faz isso para garantir a aderência à teoria, práticas e regras pelo time Scrum. O Scrum Master é um servo-líder para o time Scrum. O Scrum Master ajuda aqueles que estão fora do time Scrum a entender quais as interações com o time Scrum são úteis e quais não são. O Scrum Master ajuda todos a mudarem estas interações para maximizar o valor criado pelo time Scrum [4].

2.1.1 Etapas do Sprint

Com a caracterização do time Scrum realizada na seção 2.1 "Metodologia Ágil: SCRUM", é necessário também conhecer o processo de projeto do SCRUM, tais como: o Sprint, Sprint Planning Meeting, Daily Scrum, Sprint Review e Sprint Review Meeting [3].

De acordo com Ken Schwaber [3], o Sprint é definido como um ciclo de desenvolvimento curto em que o time foca em atingir o objetivo proposto pelo Product Owner. Durante este período o time deve ter total autoridade para seu gerenciamento, não devendo sofrer interferências externas do Product Owner ou mesmo de outras pessoas. O Sprint deve ter um tempo de execução de duas a quatro semanas. Esse tempo tem como objetivo fazer com que as funcionalidades sejam entregues e validadas pelo Product Owner [3].

O Sprint Planning Meeting é a reunião realizada na qual devem estar presentes o Product Owner, o Scrum Master e todo a equipe do Scrum. O Product Owner debate o objetivo que a Sprint deve realizar e os itens de Backlog do Produto que, se completados na Sprint, atingirão o objetivo do Sprint. Todo o time Scrum colabora com o entendimento do trabalho do Sprint [3].

A entrada da reunião de planejamento do Sprint é o Backlog do Produto, o mais recente incremento do produto, a capacidade projetada do time de desenvolvimento durante a Sprint e o desempenho passado do time de desenvolvimento. O número de itens selecionados do Backlog do Produto para a Sprint é o único trabalho do time de desenvolvimento. Somente o time de desenvolvimento pode avaliar o que pode ser completado ao longo da próxima Sprint [4].

Após o time de desenvolvimento prever os itens de Backlog do Produto que irá entregar na Sprint, o time Scrum determina a meta do Sprint, ou seja, o objetivo que será conhecido dentro do Sprint através da implementação do Backlog do Produto, fornecendo a orientação para o time de desenvolvimento sobre o porquê dele estar construindo o incremento [4].

Daily Scrum é uma reunião diária que acontece precisamente no tempo 15 minutos, para que o time de desenvolvimento possa sincronizar as atividades e criar um plano para as próximas 24 horas. Esta reunião é feita para inspecionar o trabalho desde a última Reunião Diária e prever o trabalho que deverá ser feito antes da próxima Reunião Diária. O Scrum Master assegura que o time de desenvolvimento tenha a reunião, mas o time de desenvolvimento é responsável por conduzir a Reunião Diária.[4].

Durante a reunião, os membros do time de desenvolvimento esclarecem:

- O que eu fiz ontem que ajudou o time de desenvolvimento a atender a meta do Sprint?
- O que eu farei hoje para ajudar o time de desenvolvimento atender a meta do Sprint?
- Eu vejo algum obstáculo que impeça a mim ou o time de desenvolvimento no atendimento da meta do Sprint?

A Revisão do Sprint é executada no final do Sprint para inspecionar o incremento e adaptar o Backlog do Produto se necessário. Durante a reunião de Revisão do Sprint, o time Scrum e as partes interessadas colaboram sobre o que foi feito na Sprint. Com base nisso e em qualquer mudança no Backlog do Produto durante a Sprint, os participantes colaboram nas próximas coisas que podem ser feitas para otimizar valor. Esta é uma reunião informal e a apresentação do incremento destina-se a motivar e obter comentários e promover a colaboração [4].

A Retrospectiva do Sprint ocorre depois da Revisão do Sprint e antes da reunião de planejamento da próxima Sprint. Esta é uma reunião com tempo de três horas para um Sprint de um mês. Para Sprint menores, este evento é usualmente menor. O Scrum Master garante que o evento ocorra e que os participantes entendam seu propósito. O Scrum Master participa da reunião como um membro auxiliar do time devido a sua responsabilidade pelo processo Scrum [4].

O Scrum Master encoraja o time Scrum a melhorar o processo de desenvolvimento e as práticas para fazê-lo mais efetivo e agradável para a próxima Sprint [4].

Ao final da Retrospectiva do Sprint, o time Scrum deverá ter identificado melhorias que serão implementadas na próxima Sprint. A implementação destas melhorias na próxima Sprint é a forma de adaptação à inspeção que o time Scrum faz a si próprio. A Retrospectiva do Sprint fornece um evento dedicado e focado na inspeção e adaptação, no entanto, as melhorias podem ser adotadas a qualquer momento [4].

2.1.2 Artefatos

Os artefatos do Scrum representam o trabalho ou o valor para o fornecimento de transparência e oportunidades para inspeção e adaptação. Os artefatos definidos para o Scrum são especificamente projetados para maximizar a transparência das informações chave de modo que todos tenham o mesmo entendimento dos artefatos.Podem ser representados pelo Backlog do Produto, Backlog do Sprint e interação/incremento [4].

O Backlog do Produto é uma lista ordenada de tudo que deve ser necessário no produto e é uma origem única dos requisitos para qualquer mudança a ser feita no produto. Os itens do Backlog do Produto de ordem mais alta (topo da lista) devem ser mais claros e mais detalhados que os itens de ordem mais baixa. Estimativas mais precisas são feitas baseadas em maior clareza e maior detalhamento; quanto menor a ordem na lista, menos detalhes [4].

O Backlog do Sprint é um conjunto de itens do Backlog do Produto selecionados para a Sprint, juntamente com o plano para entregar o incremento do produto e atingir o objetivo do Sprint. O Backlog do Sprint é a previsão do time de desenvolvimento sobre qual funcionalidade estará no próximo incremento e sobre o trabalho necessário para entregar essa funcionalidade em um incremento [4].

O incremento é a soma de todos os itens do Backlog do Produto completados durante a Sprint e o valor dos incrementos de todas as Sprints anteriores. Ao final do Sprint um novo incremento deve estar concluído, o que significa que deve estar na condição utilizável e atender a definição do Time Scrum [4].

2.2 Engenharia de Requisitos

Requisito é uma condição ou capacidade necessária a um usuário para resolver um problema ou alcançar um objetivo, que deve ser possuída por um sistema ou por um componente do sistema para satisfazer um contrato, padrão, especificação ou outros documentos formalmente impostos [5].

A Engenharia de Requisitos é o ramo da Engenharia de Software que envolve as atividades relacionadas com a definição dos requisitos de software de um sistema, desenvolvidas ao longo do ciclo de vida de software [2].

Na engenharia de requisitos, o requisito representa as descrições do cliente para a construção de um sistema, com o intuito de atender e solucionar um problema. Requisitos são a base de todos os produtos de software e sua elicitação, gerenciamento e entendimento são problemas comuns a todas as metodologias de desenvolvimento de software [6].

2.2.1 Tipos de Requisitos

Os requisitos de sistemas de software são classificados em: (i) requisitos funcionais, (ii) requisitos não funcionais ou (iii) requisitos de domínio [2].

Requisitos funcionais são as declarações que o sistema deve fornecer, como o sistema deve reagir a entradas específicas e como o sistema deve se comportar em determinadas situações. Em alguns casos, os requisitos funcionais podem também estabelecer explicitamente o que o sistema não deve fazer [2].

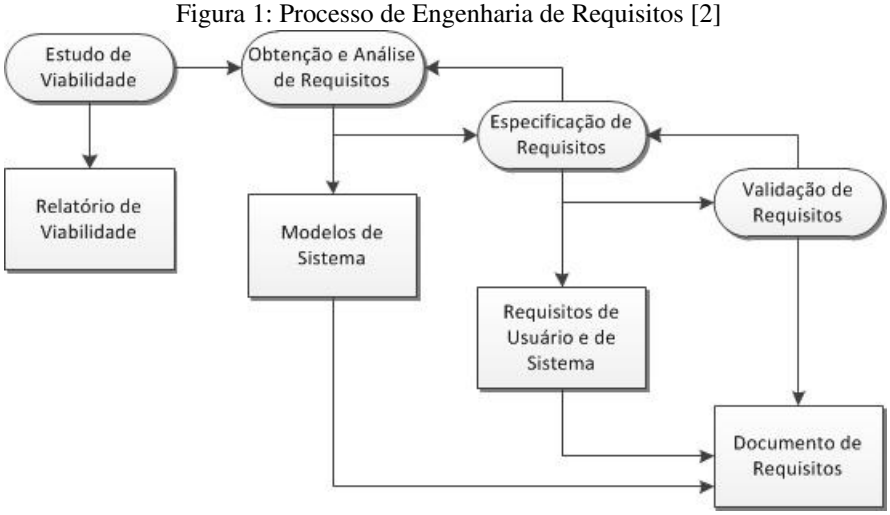
Requisitos não funcionais são restrições sobre os serviços ou as funções oferecidas pelo sistema. Eles incluem restrições de timing, restrições sobre o processo de desenvolvimento e padrões. Os requisitos não funcionais aplicam-se, frequentemente, ao sistema como um todo. Em geral eles não se aplicam às características ou serviços individuais de sistemas [2].

Requisitos de domínio são requisitos provenientes do domínio da aplicação do sistema e que refletem as características e as restrições desse domínio. Podem ser classificados como requisitos funcionais ou não funcionais [2].

2.2.2 Processo de Engenharia de Requisitos

A engenharia de requisitos é uma atividade responsável por criar e manter Documentos de Especificações de Requisitos de Sistemas (DERS). Esse processo é divido em quatro subprocessos: (i) estudo de viabilidade, (ii) elicitação e análise, (iii) especificação e (iv) validação A Figura 1 mostra o processo de engenharia de requisitos. [2].

Figura 1: Processo de Engenharia de Requisitos [2]



Estudo de viabilidade: verificação se as necessidades dos usuários podem ser satisfeitas por meio das tecnologias de software e hardware. O estudo considera se o software pode ser desenvolvido dentro de restrições de orçamento existentes [2]

Elicitação e análise: derivação de requisitos de sistema através da observação de sistemas existentes, discussões com usuários potencias compradores etc. Isso pode envolver o desenvolvimento de um ou mais modelos de sistemas e protótipos. Esta atividade ajuda o analista a compreender o sistema a ser especificado[2]

Especificação: tradução das informações elicitadas em um documento que define um conjunto de requisitos. Devem ser incluídos dois tipos de requisitos nesse documento: (i) requisitos de usuário (declarações abstratas dos requerimentos de sistema para o cliente e os seus usuários finais) e (ii) requisitos de sistema (descrição mais detalhadas da funcionalidade a ser fornecida) [2]

Validação: verificação dos requisitos em relação ao realismo, consistência e abrangência. Durante esse

processo, erros no documento de requisitos são inevitavelmente descobertos. Devem então ser feitas modificações para corrigir esses problemas [2].

2.2.3 Artefato

Artefatos são produtos de trabalho finais ou intermediários produzidos e usados durante os projetos para capturar e transmitir informações do projeto. Um artefato pode ser um dos seguintes elementos [7]:

- Um documento, como Caso de Negócio ou Documento de Arquitetura de Software;
- Um modelo, como o Modelo de Casos de Uso ou o Modelo de Design;
- Um elemento do modelo, ou seja, um elemento existente em um modelo, como uma classe ou um subsistema.

O Rational Unified Process (RUP) é um processo da engenharia de software com o objetivo de assegurar a produção de software de alta qualidade dentro de prazos e orçamentos previsíveis. Derivado dos trabalhos sobre UML e do Processo Unificado de Desenvolvimento de Software, ele traz elementos de todos os modelos genéricos de processo, apoia a iteração e ilustra boas práticas de especificação e projeto [2].

O RUP descreve um conjunto de artefatos para a engenharia de requisitos que captura e apresenta informações usadas para definir os recursos necessários do sistema, conforme descrito a seguir [7]:

- **Plano de gerenciamento de requisitos:** descreve a documentação de requisitos, os tipos de requisitos e seus respectivos atributos de requisitos, especificando as informações e os mecanismos de controle que devem ser coletados e usados para avaliar, relatar e controlar mudanças nos requisitos do produto.
- **Glossário:** define os termos importantes usados no projeto.
- **Visão:** define a visão que os envolvidos têm do produto a ser desenvolvido, em termos das necessidades e características mais importantes.
- **Especificação de requisitos de software:** descreve o sistema com uma visão geral, como também os seus requisitos funcionais e não-funcionais. Fornece aos desenvolvedores as informações necessárias para o projeto e implementação, assim como para a realização dos testes e a implantação do sistema.
- **Documento de caso de uso:** descreve cada caso de uso deve conter pré-condições e pós-condições, fluxos de eventos, prioridades e relacionamento com outros casos de uso.
- **Modelo de caso de uso:** modelo das funções pretendidas do sistema e seu ambiente, e serve como um contrato estabelecido entre o cliente e os desenvolvedores. É usado como fonte de informações para atividades de análise, design e teste.
- **Protótipo da interface do usuário:** representa um protótipo do sistema, podendo ser descrito através de esboço em papel ou figuras, bitmaps feitos com uma ferramenta de desenho ou protótipo de executável interativo.

3 Engenharia de Requisitos no SCRUM

Muitos esforços têm sido empenhados em pesquisas que discutem a relação entre Engenharia de Requisitos com Metodologia Ágil. Segundo Paetsch et al. 2003, é necessário saber o que construir antes de começar o desenvolvimento do sistema, a fim de evitar um grande retrabalho. Para não acontecer esse retrabalho, quanto mais tarde os erros são descobertos, mais caro será para corrigi-los [3], sendo possível determinar um conjunto estável de requisitos antes do projeto do sistema e do começo da codificação.

Portanto, é necessário estabelecer o trabalho da Engenharia de Requisitos do ponto de vista do desenvolvimento ágil, apresentando atividades de Engenharia de requisitos na implementação Scrum e algumas recomendações para ajudar a gestão e execução das equipes de desenvolvimento. A Engenharia de Requisitos está preocupada

Tabela 1: Aplicação da Engenharia de requisitos Requisitos no Scrum [8]

Atividade da Engenharia de Requisitos	**Scrum**
Elicitação de requesitos	- *Product Owner* formula o *Product Backlog*. - Todas as partes interessadas podem participar no *Product Backlog*.
Análise de requisitos	- Reunião de refinamento do Backlog. - Product Owner prioriza o Product Backlog. - Product Owner analisa a viabilidade de requisitos.
Documentação de requistos	- Comunicação face-a-face
Validação de requisitos	- Reuniões de revisão

com a descoberta, análise, especificação e documentação dos requisitos do sistema e merece o maior cuidado, pois os problemas inseridos no sistema durante a fase de Engenharia de requisitos são os mais caros para remover [8].

A Tabela 1 expõe como ocorre a atividade de Engenharia de Requisitos na implementação do Scrum, de forma que os papéis e eventos do Scrum são empregados em cada fase da Engenharia de uma maneira ágil.

As necessidades dos requisitos podem evoluir com o tempo, o cliente pode mudar de opinião ou o ambiente técnico e socioeconômico global pode evoluir. Todavia, a metodologia ágil requer um alto nível de interação entre clientes, gerentes e desenvolvedores. Papéis e responsabilidades dos clientes, gerentes e desenvolvedores são de suma importância e tem um amplo impacto sobre a evolução de um software projeto [9].

A seguir será apresentado algumas técnicas de captura de requisitos de uma forma ágil, utilizando métodos, como entrevistas, casos de uso, brainstorming, prototipagem, entre outras [8] [10].

Entrevistas

"Entrevistar é um método para descobrir fatos e opiniões detidos por potenciais interessados do sistema em desenvolvimento"[3].

Existem dois tipos de entrevistas: (i) entrevistas fechadas, onde um conjunto predefinido de perguntas são respondidas, e (ii) entrevistas abertas, onde não há nenhuma agenda pré-definida e uma série de questões são exploradas com as partes interessadas. Na verdade, as entrevistas são bons para a obtenção de um entendimento geral do que os stakeholders fazem e como eles podem interagir com o sistema, mas eles não são bons para o domínio entendimento requisitos. Os métodos ágeis afirmam que as entrevistas são uma maneira eficiente de se comunicar com os clientes e de aumentar a confiança entre os dois lados [8].

Casos de uso

Casos de uso descrevem as interações entre usuários e sistema, incidindo sobre o que os usuários precisam fazer com o sistema. Um caso de uso especifica uma sequência de interação entre o sistema e um agente externo (por exemplo, uma pessoa, uma peça de hardware, software de outro produto), incluindo variantes e extensões, que o sistema pode realizar. Os casos de uso representam requisitos funcionais do sistema de software e podem ser utilizados durante as fases iniciais do processo de desenvolvimento. Os analistas e os clientes devem examinar cada caso de uso proposto para validá-lo [10].

Esta é uma técnica baseada em cenário, ou seja, exemplos de interação em que um único tipo de interação entre o usuário e o sistema é simulado. Cenários devem incluir uma descrição do estado do sistema antes de entrar e após a conclusão do cenário, quais as atividades podem ser simultâneas, o fluxo normal de eventos e exceções aos acontecimentos [10].

Brainstorming

É uma técnica de grupo para gerar novas ideias úteis e promover o pensamento criativo. Brainstorming pode ser usado para obter novas ideias e recursos para a aplicação, definir o projeto ou problema para trabalhar e para diagnosticar problemas em um curto espaço de tempo. O gerente de projeto desempenha um papel importante, determinando o tempo da sessão, certificando-se de que não haja discussões crescentes sobre determinados temas,

e de que cada corpo expressa sua opinião livremente. Após a finalização da sessão, os tópicos são avaliados pela equipe. Além disso, as conexões e dependências entre as ideias discutidas são representados através da visualização de gráficos, por exemplo, de modo que os conflitos com outros requisitos são encontrados e avaliados [8].

Prototipagem

Um protótipo de um sistema é uma versão inicial do sistema, que está disponível no início do processo de desenvolvimento. Protótipos de sistemas de software são muitas vezes utilizados para ajudar a obter e validar requisitos do sistema. Existem dois diferentes tipos de protótipos: (i) o protótipo descartável, que ajuda a descobrir requisitos que causam dificuldades de compreensão, e (ii) o protótipo evolutivo, que oferece um sistema usável para o cliente e pode se tornar uma parte do sistema final [10].

Joint Application Development (JAD):

É uma oficina usada para elicitar os requisitos de negócios durante o desenvolvimento de um sistema. As sessões JAD também incluem abordagens para aumentar a participação do usuário, acelerar o desenvolvimento e melhoria da qualidade de especificações [8].

No ambiente ágil, em caso de conflitos entre os requisitos das partes interessadas, o uso de JAD pode ajudar a promover o uso de um facilitador profissional que pode ajudar a resolver conflitos. Além disso, as sessões JAD incentivam o envolvimento do cliente e a confiança no sistema desenvolvido [8].

Modelagem

Modelos do sistema são importantes para análise e o processo de design [10]. Em ambiente ágil, a modelagem é dividida em três seções: (i) modelos a serem implementados, (ii) modelos em fase de implementação, e (ii) modelos concluídos. Este esquema deve ser documentados e fornece uma representação visual do status do projeto [9].

No processo de levantamento de requisitos, nem sempre irá existir uma técnica padrão para alcançar um levantamento de requisitos desejado. Sendo, portanto, necessário conhecer diversas técnicas para saber qual a melhor a ser aplicada em cada situação.

4 Estudo de Caso

Esta seção relata uma experiência da engenharia de requisitos na metodologia ágil Scrum, descrevendo os métodos e as técnicas utilizadas em um projeto de desenvolvimento de software com metodologia ágil, citando os principais problemas encontrados e destacando a agilidade no desenvolvimento de software.

4.1 Avaliação Experimental

O estudo de caso foi realizado em uma empresa que utiliza a metodologia ágil Scrum no desenvolvimento de software. A empresa possui mais de 100 funcionários e está localizada na cidade de Rio de Janeiro – RJ, Brasil.

A avaliação foi realizada levando em considerações os seguintes aspectos: (i) os processos utilizados no desenvolvimento de software, (ii) as técnicas adotadas na engenharia de requisitos do ponto de vista ágil, (iii) os artefatos de requisitos produzidos no projeto de desenvolvimento de software, (iv) os principais problemas encontrados na engenharia de requisitos atrelados a metodologia ágil e (v) os maiores desafios da aplicação da engenharia de requisitos com metodologia ágil. A avaliação desses critérios, nos permite conhecer os pontos de dificuldade, descobrir problemas e apresentar possíveis melhorias.

4.2 Resultados e Discussão

Na engenharia de requisitos, o requisito representa as descrições do cliente para a construção de um sistema, com o intuito de atender e solucionar um problema. Requisitos são a base de todos os produtos de software e sua elicitação, gerenciamento e entendimento são problemas comuns a todas as metodologias de desenvolvimento de software [6].

A empresa utiliza processos bem definidos no desenvolvimento de software, dentre os quais se destacam: requisitos, análise e projeto, gerenciamento de mudança, gerenciamento de projeto, qualidade e teste.

Sobre as técnicas de elicitação de requisitos, descritas na seção 3, foi questionado quais as técnicas adotadas pela empresa na engenharia de requisitos no ponto de vista ágil. A empresa adota as seguintes técnicas: prototipagem, revisões de requisitos, documentação, priorização e brainstorming. Estas técnicas são consideradas imprescindíveis para a obtenção de requisitos mais precisos, proporcionando um alto nível de interação e entendimento entre as partes envolvidas no projeto (Product Owner, Scrum Master e time Scrum). Quanto maior o entendimento do projeto a nível de requisitos, menor será o risco de mudança e maior será a confiança da equipe em atender as necessidades do projeto de software solicitado pelo cliente.

Nos projetos de software é necessário que os requisitos sejam documentados, produzindo artefatos sobre requisitos funcionais, user stories, análise de projeto, entre outros. A empresa relata que os requisitos são documentados através de user stories e distribuídos em um quadro, conseguindo ajudar a planejar o projeto, priorizar as atividades, assim como distribui-las para o time Scrum. As user stories facilitam o entendimento da funcionalidade a ser desenvolvimento por cada integrante do time. No entanto, algumas vezes as user stories dificultam um pouco o processo de comunicação e desenvolvimento quando se trabalha com equipes remotas, ou seja, distribuídas geograficamente, uma vez que as histórias são curtas e não trazem muitos detalhes do requisito a ser desenvolvido.

A empresa relata que o engajamento de toda a equipe é um desafio para construir o processo Scrum, assim como para cumprir os prazos e atividades previstas pelas Sprints. Uma das características do Scrum é ter uma equipe auto-organizável e multidisciplinar, ou seja, uma equipe que organiza e gerencia seu próprio trabalho e possui as competências necessárias para completar o trabalho, respectivamente. A comunicação face-a-face entre o time Scrum, assim como a liderança e o encorajamento dado pelo Scrum Master, são pontos muito relevantes para melhorar o processo de desenvolvimento e as práticas para fazê-lo mais efetivo e agradável. A compreensão de forma clara dos requisitos e a motivação da equipe ajudam a somar confiança entre os membros da equipe, minimizando a dificuldade de engajamento de toda a equipe.

5 Conclusão

Este trabalho apresentou técnicas e abordagens da Engenharia de Requisitos utilizadas durante os processos de desenvolvimento ágil, incluindo o estudo de viabilidade, elicitação, análise, documentação e validação. Da mesma forma, expõe a metodologia ágil Scrum, definindo os papéis do Scrum, as etapas dos Sprints e os artefatos.

Para a construção do processo de engenharia de requisito de uma forma ágil, alguns aspectos são imprescindíveis, tais como: colaboração do cliente, bons desenvolvedores ágeis e gerentes de projeto experientes. Além disso, a comunicação entre os membros da equipe e as partes interessadas podem ser o segredo do sucesso na entrega final do produto, focando na qualidade, prazos, custo, agilidade e satisfação do cliente.

Como trabalhos futuros, sugere-se a realizar um comparativo da engenharia de requisitos em projetos que usam Scrum, metodologia de software tradicional e outro projeto que não usa metodologia de processo para propor melhorias no processo de requisitos para esses projetos.

Agradecimentos

A todas as pessoas que, direta ou indiretamente contribuíram com carinho e atenção durante a construção desse trabalho.

Referências

[1] THAYER, R. H.; BAILIN, S. C.; DORFMAN, M. *Software Requirements Engineerings, 2Nd Edition*. 2nd. ed. Los Alamitos, CA, USA: IEEE Computer Society Press, 1997. ISBN 0818677384.

[2] SOMMERVILLE, I. et al. *Engenharia de software*. ADDISON WESLEY BRA, 2008. ISBN 9788588639287. Disponível em: <https://books.google.com.br/books?id=ifIYOgAACAAJ>.

[3] SCHWABER, K.; BEEDLE, M. *Agile Software Development with Scrum*. 1st. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001. ISBN 0130676349.

[4] Schwaber, K.; Sutherland, J. Guia do scrum. um guia definitivo para o scrum: As regras do jogo., 2013.

[5] IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*, p. 1–84, Dec 1990.

[6] AURUM, A.; WOHLIN, C. *Engineering and Managing Software Requirements*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005. ISBN 3540250433.

[7] CORPORATION, R. S. Rational unified process: Artefatos, 1987 - 2001.

[8] LUCIA A.; QUSEF, A. Requirements engineering in agile software development. In: . [S.l.]: Journal of Emerging Technologies in Web Intelligence, 2010.

[9] SILLITTI, A.; SUCCI, G. Engineering and managing software requirements. In: ____. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. cap. Requirements Engineering for Agile Methods, p. 309–326. ISBN 978-3-540-28244-0. Disponível em: <http://dx.doi.org/10.1007/3-540-28244-0_14>.

[10] PAETSCH, F.; EBERLEIN, A.; MAURER, F. Requirements engineering and agile software development. In: *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on*. [S.l.: s.n.], 2003. p. 308–313. ISSN 1080-1383.

Uma Reengenharia em Mapas do Software para Visualização de Dados Datawrapper

Jefferson L. S. Ferreira[1]
Ed P. Bezerra [1]

Resumo: A necessidade de mostrar dados complexos de forma mais compreensível aumenta a cada dia. Graças à tecnologia atual, há ferramentas para transformar esses dados complexos em infográficos interativos. Um tipo de infográfico bastante utilizado nos dias atuais, que consegue expor uma grande quantidade de informação de forma clara e sucinta, é o mapa. Quando infográficos interativos na forma de mapas são usados, o entendimento de dados complexos se torna mais fácil. Muitas ferramentas são capazes de gerar tais tipos de infográficos. Entretanto, algumas limitações dessas ferramentas para geração de infográficos interativos atrapalham o entendimento de alguns dados mais complexos. O Datawrapper possibilita a criação de infográficos com mapas, mas restringe tal funcionalidade a mapas de países e estados, impossibilitando a criação de mapas com municípios. Aprimorar o Datawrapper para gerar as 27 unidades federativas do Brasil foi o objetivo dessa pesquisa. Um estudo de caso para gerar o mapa do estado da Paraíba com seus 223 municípios foi realizado. Este artigo aborda esta problemática e a solução adotada.

Palavras-chave: Datawrapper. Infografia. Infografia interativa. Mapas. Webjornalismo.

Abstract: *The necessity to display complexes data more comprehensive form is more important every day. Thanks to today's technology, there are tools to transform this complexes data to interactive infographics. A type of infographics often used today that can display a big quantity of information clearly and comprehensive, is the map. When interactive infographics in the map form are used, the understanding of complexes data becomes easier. Many tools are able to generate this infographics type. However, some limitations of these tools to generate interactive infographics hinder the understanding of some more complex data. The Datawrapper allow the creation of infographics with maps, but limits this function to states and countries maps, make impossible the creation of counties maps. Improve Datawrapper to generate the 27 federative units of Brazil was the objective of this research. We did a case study to generate the state map of Paraiba with its 223 counties. This article discusses this problem and the solution adopted.*

Keywords: Datawrapper. Infography. Interactive Infography. Maps. Webjournalism.

1 Introdução

A representação de dados se torna cada dia mais importante no dia a dia da maioria das pessoas. À medida que os dados se tornam mais complexos, mais difícil se torna seu entendimento. Isto nos obriga a buscar formas que facilitem sua compreensão. Uma das maneiras de expressar dados complexos é utilizando ferramentas de infografia interativa. Tais ferramentas nos permitem exibir informações de grandes bases de dados com mais clareza do que em formato de tabela ou qualquer outro tipo de infográfico estático, pois dispõem de mecanismos capazes de amostrar dados em diversos formatos de infográficos.

A infografia, propriamente dita, busca traduzir a informação, em geral complexa, tornando-a de fácil entendimento. De acordo com o aperfeiçoamento de novas técnicas de produção, as possibilidades da infografia foram ampliadas e potencializadas, tanto para o jornalismo impresso, quanto para o webjornalismo. Ou seja, "a ênfase em conceitos como a não-linearidade, a utilização de objetos visuais, confirmando a preocupação cognitiva aliada ao projeto de interface, reforça o caráter comunicacional das novas mídias"[1].

Quando levamos em consideração a exibição de dados geográficos é possível notar a ausência de ferramentas que possibilitem a geração de infográficos interativos utilizando mapas. A maioria das opções não

[1] Centro de Informática da UFPB, Unidade V, João Pessoa (PB) - Brasil
{jefferson.lacerda@eng.ci.ufpb.br,edporto@di.ufpb.br}

dispõe dessa funcionalidade ou é bastante limitada. Por exemplo, a ferramenta Tableau[1] permite a marcação de municípios com mais de 15.000 habitantes, já que ela usa a API geonames[3] como fonte de informações para seus mapas. Tal limitação abre margem para o desenvolvimento de novas ferramentas ou aprimoramento de ferramentas existentes.

A disponibilização de uma ferramenta para exibir dados referentes a municípios traria inúmeras vantagens para usuários finais interessados em manipular dados em forma de mapa. Dados esses que podem conter informações sobre educação, saúde ou segurança, mas são de difícil compreensão se não expostos da maneira correta.

Neste artigo procuramos discutir aprimoramentos feitos em uma ferramenta de geração de infográficos interativos, chamada de Datawrapper, para exibir dados referentes a municípios, independentemente da quantidade de seus habitantes. Após a modificação, se tornou possível gerar infográficos interativos usando o mapa do Estado da Paraíba com todos os seus municípios.

Abaixo veremos uma breve motivação da pesquisa, que foi a necessidade de disponibilização de uma ferramenta capaz de gerar infográficos interativos em forma de mapas para estados brasileiros. Após a motivação, 4 sessões serão apresentadas: Ferramentas para Visualização de Dados, onde é feita uma comparação entre o Datawrapper e o Tableau; Uma Reengenharia para o Datawrapper, que contém a parte do desenvolvimento do trabalho; Datawrapper Mapas, demonstrando os resultados; e Considerações finais do trabalho.

1.1. Motivação

No nosso estudo constatamos que municípios brasileiros com menos de 15 mil habitantes não são exibidos por ferramentas para visualização de dados tais como Tableau. Por exemplo, um mapa interativo dos municípios paraibanos não apresenta todos os seus 227 municípios, mas apenas aqueles com mais de 15 mil habitantes. A Figura 1 ilustra o mapa do Estado da Paraíba, gerado pela ferramenta Tableau, a partir de uma base de dados referentes a investimentos do governo federal através do Programa de Aceleração do Crescimento(PAC).

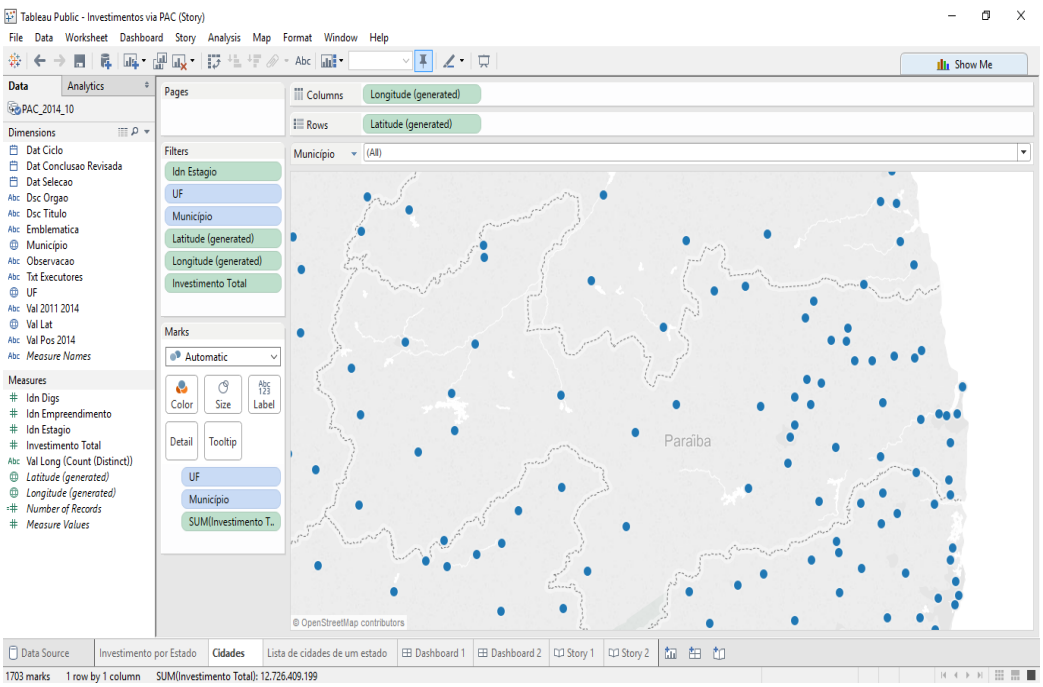


Figura 1: Mapa da Paraíba no Tableau - apenas municípios com mais de 15.000 habitantes marcados

Percebemos que o Tableau exibe apenas as cidades com mais de 15.000 habitantes. Ao tentar filtrar a cidade de Algodão de Jandaíra, que possui 2.475 habitantes, o mapa paraibano se torna oculto (Figura 2) devido à impossibilidade do Tableau marcar municípios com menos de 15.000 habitantes.

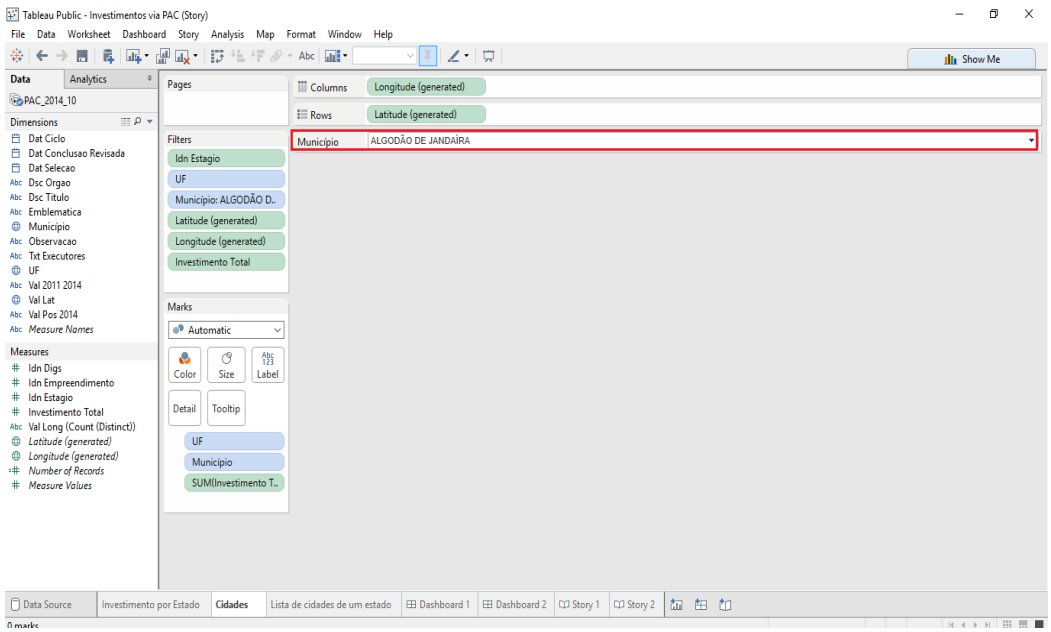


Figura 2: Tentativa de filtragem por município

Percebemos que ao filtrar a consulta para o município de Algodão de Jandaíra, não aparece o mapa desta cidade porque a mesma possui menos de 15.000 habitantes. Ademais nenhuma mensagem sobre a ausência de resultado é emitida.

2 Ferramentas para Visualização de Dados

De acordo com [4], "A visualização de dados é a representação gráfica de informações abstratas para dois propósitos: sentido de decisão (também chamada de análise de dados) e de comunicação". Dados são informações que carregam consigo uma história importante. Visualização de dados é um meio eficiente para descobrir e entender essas histórias, que permite apresentá-las numa linguagem mais comunicativa. Por exemplo, informações estatísticas são abstratas, pois não descrevem objetos físicos. Praticamente qualquer coisa pode ser descrita visualmente, mesmo não pertencendo ao mundo físico. Seja vendas de produtos, incidência de uma doença ou desempenho de atletas em um esporte. Todavia, é necessário encontrar uma maneira de se dar forma para tais informações. Para visualizar dados de forma eficaz, devemos entender princípios que são derivados a partir de um entendimento de percepção humana.

A visualização de dados muda o equilíbrio entre a percepção e cognição para tirar melhor partido das habilidades do cérebro. Ver algo (percepção visual) é extremamente rápido e eficiente. Vemos imediatamente, com pouco esforço. Pensar algo (cognição) é muito mais lento e menos eficiente que ver algo, ou seja, a visualização de dados muda o equilíbrio para uma maior utilização da percepção visual, aproveitando nossos olhos sempre que possível e aprimorando a capacidade de cognição de uma determinada história complexa [5].

Gráficos são representações visuais usadas para codificar dados, conceitos, conexões e localizações geográficas. Todos os gráficos apresentam dados e permitem a exploração destes dados. De acordo com [5], "alguns gráficos são quase totalmente apresentação", ou seja, eles permitem apenas uma quantidade limitada de exploração. Por isso, diz-se que estes são mais infográficos do que apenas visualização. O objetivo de um infográfico é narrar uma história complexa num espaço limitado. Logo, um infográfico deve facilitar a análise do que é mostrado. A Figura 3 apresenta um infográfico sobre a votação de cada banca da Câmara dos Deputados no processo de impeachment da presidenta da República do Brasil.

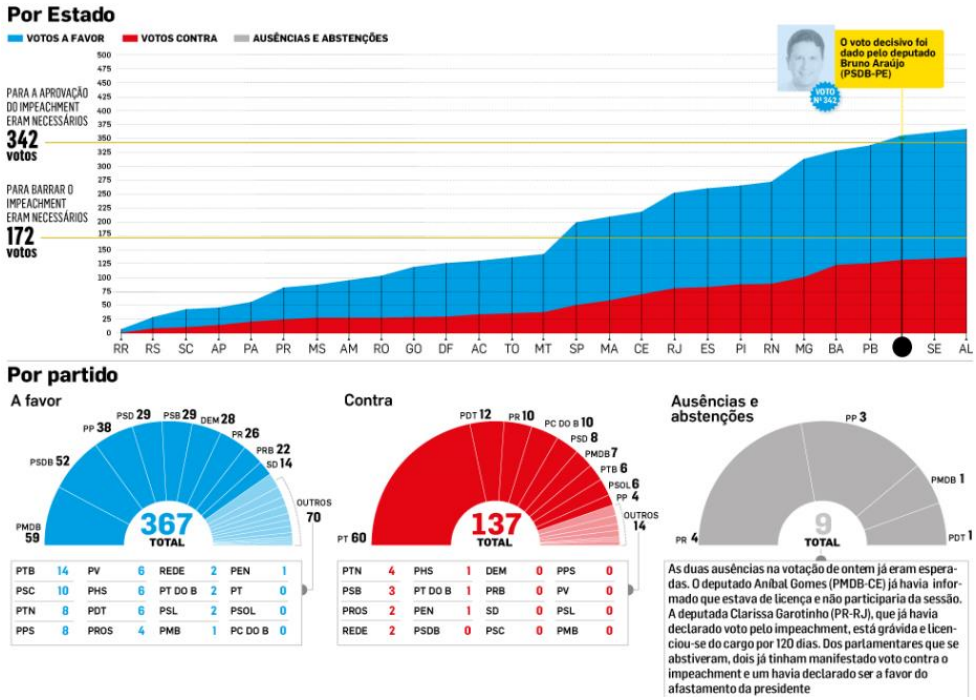


Figura 3: Como votaram as bancadas - Fonte: http://www.estadao.com.br/infograficos/politica,como-votaram-as-bancadas,571262

Há um destaque para o voto de número 342, que é a quantidade requerida de votos para abertura do processo de impeachment.

A infografia interativa nada mais é do que uma área da infografia que representa infográficos comuns com a adição de interatividade com o leitor. Interatividade essa possibilitada pelas mídias interativas, como internet, por exemplo, onde é possível hospedar uma página web que contenha um infográfico onde o usuário possa aplicar filtros de informações e modificar dinamicamente a visualização atual. Alguns exemplos de infográficos interativos podem ser visualizados através do site https://jeffersonlac.wix.com/infolab[7].

2.1 Tableau

O Tableau é um software gerador de infográficos interativos baseado em bancos de dados que funciona a partir de uma aplicação local e instalável no computador do usuário. Disponível para Windows, ele permite a geração de infográficos interativos com 24 formas diferentes e que são sugeridas de acordo com o conjunto de dados disponibilizado pelo usuário. A geração de mapas no Tableau se sai muito bem e pode ser considerada a melhor entre as ferramentas do mesmo gênero. Entretanto, há a limitação de região: suporta apenas municípios com mais de 15.000 habitantes. Além disso, é uma ferramenta de código fechado o que impossibilita qualquer aprimoramento. Seus usuários devem pagar para usar o Tableau, com exceção do Tableau Public, uma versão destinada apenas a consulta de estudantes e pesquisadores.

2.2 Datawrapper

O Datawrapper é uma ferramenta de visualização on-line, que permite aos usuários criarem infográficos interativos em forma de barras, colunas, pizza, rosquinha, mapas, etc. A ferramenta é projetada para ser acessível a qualquer usuário de qualquer área de interesse que precise visualizar dados com mais clareza e contexto sem perder tanto tempo com configurações e funcionalidades desnecessárias. O Datawrapper também se destaca por ser uma ferramenta de código aberto, o que permite que qualquer usuário do tipo desenvolvedor realize modificações na ferramenta original, possibilitando o incremento nas funcionalidades já existentes.

O Datawrapper possui algumas limitações como as seguintes:

1. O tratamento de dados, já que o Datawrapper não possui praticamente nenhuma ferramenta voltada a esse fim, obrigando os usuários a tratarem seus dados em uma outra ferramenta localmente e carregarem os dados praticamente prontos para o uso;
2. Geração de mapas limitado a demarcação de países e estados;
3. Impossibilidade de lidar com grandes quantidades de dados.

A ferramenta original está disponível publicamente através do link https://datawrapper.de/ e possui três versões: Basic, Single e Team. Entretanto, apenas a Basic está disponível gratuitamente. A Figura 4 apresenta a tela principal do Datawrapper Basic.

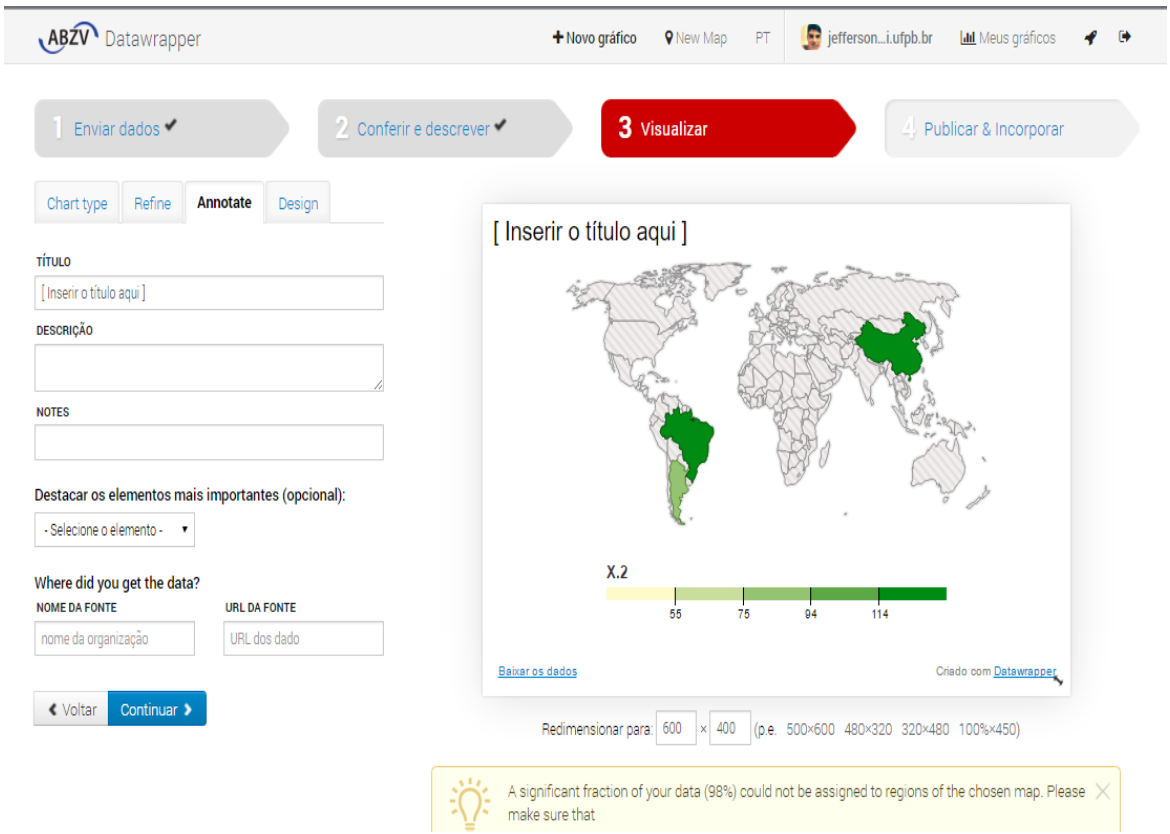


Figura 4: Datawrapper basic

Percebemos a tela de personalização de um mapa com a opção 3 (Visualizar) em destaque.

3 Uma Reengenharia para o Datawrapper

O Datawrapper só permite a visualização de países com suas respectivas unidades federativas, excluindo a possibilidade de marcação de áreas menores, como as cidades de uma dessas unidades federativas. Nossa alteração da ferramenta visa possibilitar a visualização de mapas, incluindo seus municípios. Um estudo de caso foi implementado com estados brasileiros e suas respectivas cidades.

Nossa metodologia possui três etapas principais: o estudo do funcionamento do Datawrapper, a preparação de dependências e a organização dos arquivos. A Figura 5 ilustra 10 etapas e subetapas para modificação da ferramenta Datawrapper. As subseções 3.1, 3.2 e 3.3 detalham as 3 etapas principais (etapas 1, 9 e 10) com respectivas subetapas.

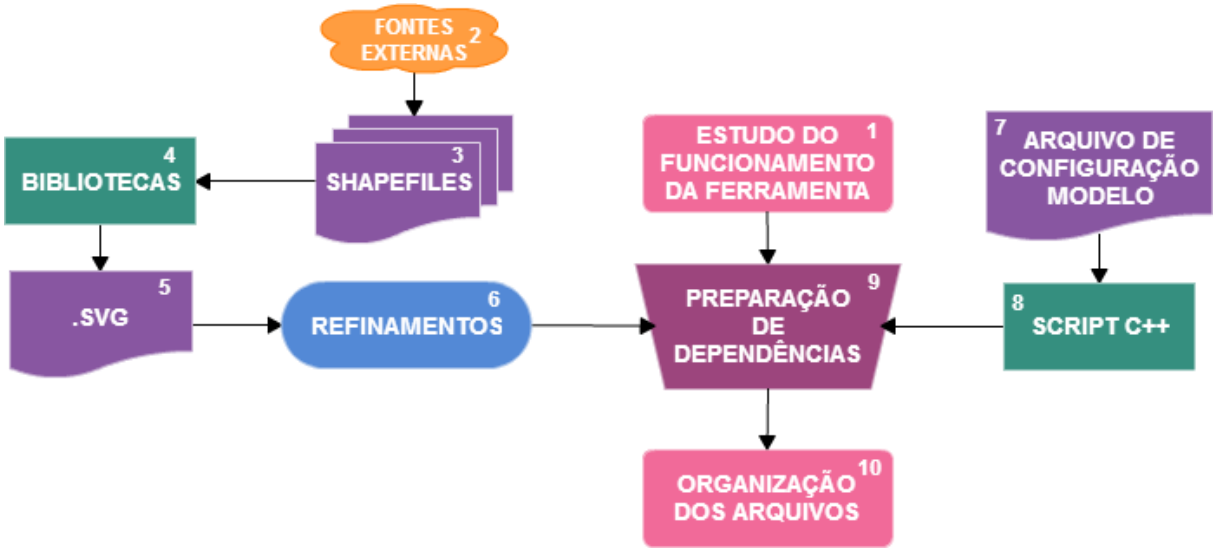


Figura 5: Diagrama de Etapas da Reengenharia

3.1 Estudo do funcionamento da ferramenta

Para adicionar mais funcionalidades à geração de mapas da ferramenta, foi necessário recriá-la com a adição de outros mapas. Uma análise de como o Datawrapper funcionava (etapa 1) foi realizada a fim de definir etapas da reengenharia: quais arquivos eram necessários para gerar o mapa, como o Datawrapper reconhecia os arquivos e como os mesmos deveriam estar organizados para que o infográfico interativo final pudesse ser criado. Ao final da etapa 1, percebemos que o Datawrapper depende de três arquivos (um vetor SVG e dois arquivos de configuração nomeados de map.json e locale.json). Eles são criados a partir de padrões estabelecidos pelos desenvolvedores da ferramenta (subseção 3.2) e devem estar organizados de uma maneira particular no diretório do Datawrapper (subseção 3.3).

3.2 Preparação de dependências

É possível ver que há vários passos a serem concluídos antes de passar da etapa 9 do diagrama. Um vetor do tipo SVG (Scalable Vetor Graphics) é um arquivo que usa a linguagem XML para descrever de forma vetorial desenhos e gráficos bidimensionais de forma estática ou dinâmica. Nessa pesquisa, os SVG são usados em forma de mapas para exibição ao usuário. Os map.json e o locale.json são arquivos de configurações contendo as etiquetas (label) usadas pela aplicação para identificar e modificar dinamicamente a forma atual de uma área do vetor SVG. A partir das informações destes arquivos o Datawrapper reconhece, por exemplo, qual a porção do mapa que deve colorir ou deixar sem cor.

Um outro arquivo importante na criação dos mapas é o shapefile. Shapefile é um arquivo de armazenagem de dados vetoriais da Esri (empresa americana especializada na produção de soluções para a área de informações geográficas) para armazenar a posição, o formato e os atributos de feições geográficas. É armazenado como um conjunto de arquivos relacionados e contém uma classe de feição[8].

A biblioteca Kartgraph.py gera vetores do tipo SVG a partir de arquivos de informações geográficas, os shapefiles (etapas 2, 3 e 4) de onde retiram as delimitações territoriais e informações gerais (como nome do estado/cidade) para transformarem no mapa gráfico. Ao final da etapa 9 teremos um mapa armazenado no arquivo vetorial .SVG (etapa 5). Para ilustrar o processo de criação do arquivo vetorial SVG geramos o mapa do estado da Paraíba (Figura 6).

Figura 6: Mapa do estado da Paraíba gerado pela biblioteca Kartograph

Os shapefiles utilizados nessa pesquisa foram encontrados em [9] (informações sobre o estado da Paraíba). Na fase de criação dos SVG, foi encontrado um problema com a resolução do arquivo. O arquivo resultante (Figura 6) se tornou muito grande graças ao alto nível de detalhe, o que exigia mais processamento durante a execução da ferramenta para demarcar as áreas que o usuário desejava, tornando-a lenta e impossibilitando seu uso. Para resolver tal problema, realizamos um refinamento (etapa 6) através da própria biblioteca Kartograph.py que permitiu diminuir a resolução, e consequentemente o tamanho do arquivo resultante. Isso aumentou o desempenho da geração do infográfico, sem perda de qualidade do mapa. O arquivo de mapa resultante, que antes possuía 1,9MB, passou a ter apenas 241KB.

Com posse dos SVG corretamente simplificados, foi necessário a criação dos arquivos de configurações (map.json e locale.json) que o Datawrapper exige para geração do infográfico interativo. Para criarmos tais arquivos, seguimos os padrões dos já existentes. Foi utilizado um pequeno script em C/C++ (etapa 8) que introduzia o nome das demarcações geográficas nas tag de um arquivo-padrão (map.json ou locale.json) tido como modelo (etapa 7). O script escrito é genérico e possibilita, como descrito acima, a geração de um arquivo de configuração para qualquer mapa em SVG gerado com a biblioteca Kartograph.py. O Datawrapper identifica a área a ser destacada do restante a partir de *label* (etiquetas) no vetor SVG. No caso dos estados brasileiros, usamos como *label* a sigla de cada estado. Para marcar o estado de Pernambuco, por exemplo, deve-se usar o *label* PE. Já no caso do mapa dos municípios paraibanos, como não existem siglas bem definidas que os identifique, foram usados o nome por extenso de cada cidade. Se a intenção for marcar a cidade de João Pessoa, o *label* a ser usado será “João Pessoa”.

3.3 Organização dos arquivos

Após a preparação de todos os arquivos necessários, chegamos à etapa número 10 na qual devemos organizar os arquivos obtidos no diretório de instalação do Datawrapper de acordo com a estrutura usada para que a ferramenta reconheça o novo mapa.

Os arquivos criados na etapa 9 foram o vetor SVG, o map.json e o locale.json. Para que sejam acessíveis pelo Datawrapper, eles devem estar numa pasta nomeada com o título do mapa, que por sua vez fica dentro da pasta do plug-in de visualização de mapas contido na instalação do Datawrapper. A ilustra a localização dos arquivos.

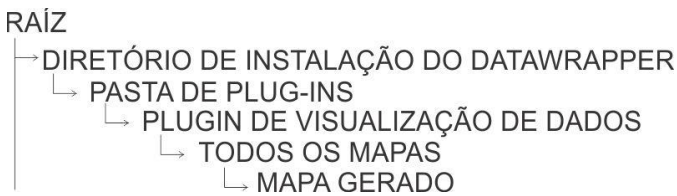


Figura 7 - Diretório dos arquivos de configuração do mapa

Os três arquivos estão armazenados na pasta MAPA GERADO. Concluídas as 10 etapas, chegamos à aplicação modificada e pronta para o usuário final. A seção 4 detalhará a ferramenta resultante, chamada Datawrapper Mapas, após a reengenharia.

4 Datawrapper Mapas

Hospedamos a ferramenta resultante no portal datavis.com.br:8090 de forma a disponibilizá-la para o público em geral. A ferramenta atualizada permite a criação de infográficos interativos na forma de mapas para os estados brasileiros e para os municípios da Paraíba em específico, além de todas as outras funcionalidades que já eram oferecidas pela ferramenta antes da modificação realizada. A Figura 8 apresenta a tela principal do Datawrapper Mapas.

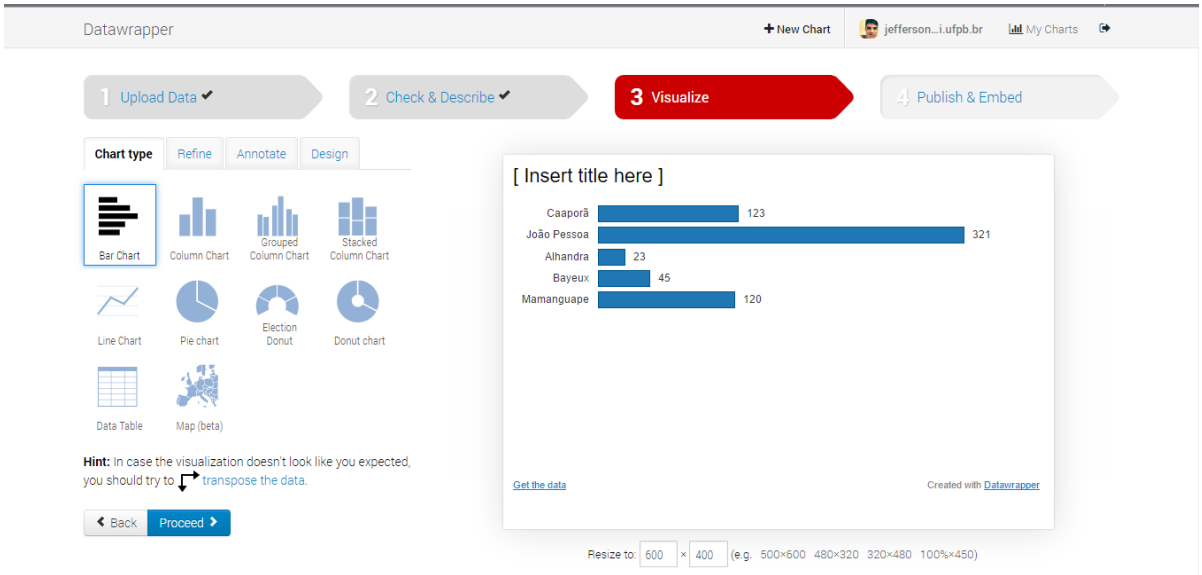


Figura 8 – Tela Principal do Datawrapper Mapas

Percebemos um infográfico em forma de barras horizontais destacando alguns municípios paraibanos e dados aleatórios, usados apenas para ilustrar a geração de um infográfico em forma de mapas. A Figura 9 ilustra o mapa da Paraíba com seus municípios, destacando 5 municípios paraibanos.

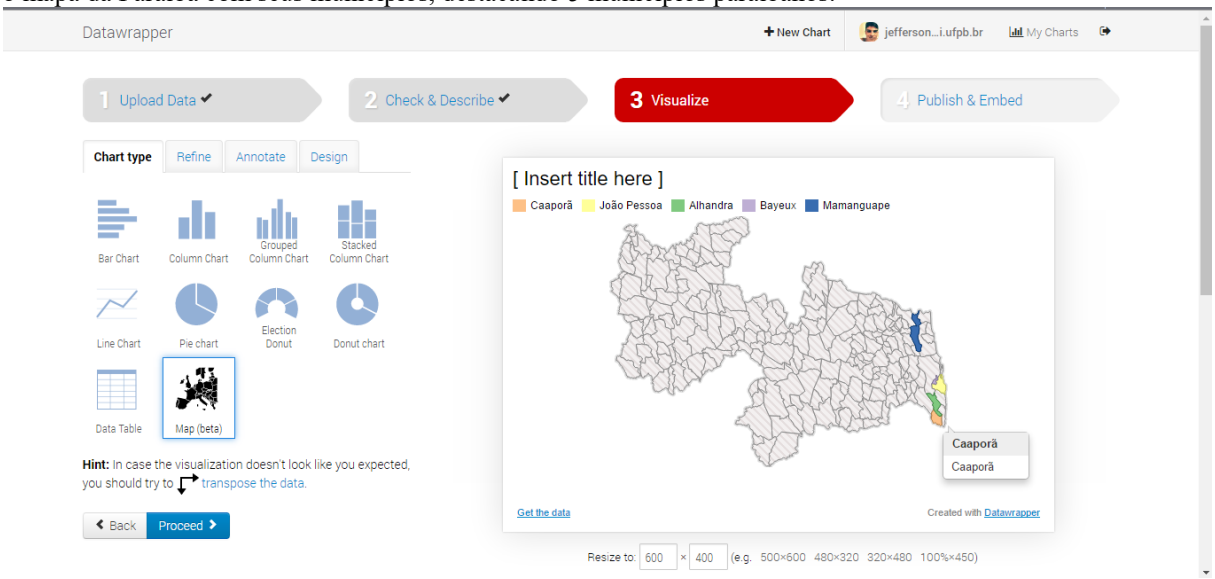


Figura 9 - Mapa dos municípios paraibanos

O mapa mostra 5 municípios paraibanos, com o ponteiro do mouse em cima do município de Caaporã, exibindo, assim, o *tooltip* referente à cidade de Caaporã.

5 Considerações finais

Após levantamento bibliográfico, não encontramos qualquer ferramenta online de geração de infográficos interativos que possibilitasse a geração de mapas interativos de áreas regionais. Isto limita possibilidades de consulta para usuários interessados em mapas.

Uma das ferramentas que possibilitam algo parecido com a alteração que nós fizemos no Datawrapper é o Tableau. Entretanto, tal ferramenta é paga para o público em geral e não possui acesso online (direto pelo navegador web), mas sim é uma aplicação que deve ser baixada e instalada (aplicação desktop), o que não significa praticidade na hora da geração de infográficos. Além disso, o Tableau é bem completa e, consequentemente, complexa de se usar.

A alteração da ferramenta Datawrapper abriu possibilidades na geração de infográficos interativos na forma de mapa. A partir desta modificação, usuários finais têm gratuitamente uma ferramenta online que possibilita gerar infográficos interativos e dinâmicos a partir de bancos de dados com informações municipais, uma vez que após a modificação realizada na ferramenta, o Datawrapper passou a possibilitar a geração de infográficos em forma de mapa para municípios, sem restrição quanto à quantidade de habitantes, limitação contida no Tableau, que exibe apenas municípios com mais de 15.000 habitantes. Isto facilita a consulta de dados em nível municipal em mapas. O usuário final precisa seguir os seguintes passos para iniciar e finalizar a criação de infográficos interativos: realizar um cadastro, subir sua base de dados (upload), e iniciar a edição do mapa. Ademais, com a publicação do Datawrapper Mapas, através do endereço datavis.com.br:8090, é possível motivar outras pessoas a implementarem suas próprias modificações. Como trabalho futuro disponibilizaremos o mapa de todos os municípios brasileiros, separados por seus estados.

Agradecimentos

Agradecemos à CNPq por todo apoio prestado e à UFPB por seu apoio estrutural e confiança no trabalho desenvolvido.

Referências

[1] MELLO, Paulo Cezar Barbosa. Cotidiano tecnologicamente criativo: internet, multimídia, hipermídia. IN: BERTOMEU, João Vicente Cegato (Org.). Criação visual e multimídia. São Paulo: Cengage Learning, 2010.

[2] TABLEAU. Disponível em: <http://www.tableau.com/pt-br/mapdata>. Acesso em: 14 abril 2016.

[3] GEONAMES. Disponível em: <http://www.geonames.org/>. Acesso em: 13 jan. 2016.

[4] FEW, Stephen (2013): Data Visualization for Human Perception. In: Soegaard, Mads and Dam, Rikke Friis (eds.). "The Encyclopedia of Human-Computer Interaction, 2nd Ed.". Aarhus, Denmark: The Interaction Design Foundation. Available online at http://www.interaction-design.org/encyclopedia/data_visualization_for_human_perception.html>. Acesso em: 16 nov. 2015.

[5] PEDROZA, Natan; BEZERRA, Ed Porto; NICOLAU, Marcos. Os Recursos da Infografia na Prática do Webjornalismo. XXXVI Congresso brasileiro de Ciências da Comunicação, Manaus, 2103.

[6] CAIRO, Alberto. The Functional Art: an introduction to information Graphics and visualization. New Readers, 2013.

[7] INFOLAB. Disponível em: <https://jeffersonlac.wix.com/infolab>. Acesso em: 04 jun. 2016.

[8] SHAPEFILES. Disponível em: <https://doc.arcgis.com/pt-br/arcgis-online/reference/shapefiles.htm>. Acesso em: 14 abril 2016.

[9] AESA. Disponível em: <http://www.aesa.pb.gov.br/>. Acesso em: 02 dez. 2015.

Uso do modelo *PMO Maturity Cube* no diagnóstico da aderência do processo de gerenciamento de portfólio de projetos do Nível F do MR-MPS-SW

[1] Fábio Henrique F. de Sousa
[2] Adriano Bessa Albuquerque

Resumo. *A concorrência exigi cada vez mais que a organização trabalhe de forma estratégica e que os investimentos realizados na execução dos projetos sejam gerenciados e controlados de forma otimizada. Para se ter resultados que atendam as expectativas, as organizações vem adotando abordagens no que se referem a implementação de modelos de maturidade de gerenciamento de portfólio de projetos. Esses modelos visam a maximização dos benefícios de resultados dos projetos e da alocação de recursos pela organização. No meio deste contexto, aparecem os Escritórios de Gerenciamento de Projetos - EGP, protagonistas em todo o processo. Este artigo tem como objetivo apresentar os resultados da aplicação do modelo de maturidade chamado PMO Maturity Cube para identificação do nível de aderência da organização nas exigências de atendimento dos resultados esperados do processo de Gerenciamento de Portfolio de Projetos do Nível F do MR-MPS-SW.*

Palavras-chave: Gestão de Portfólio de Projetos, MR-MPS-SW, *PMO Maturity Cube*.

Abstract. *Competition increasingly demanded that the organization works strategically and that investments in the implementation of projects to be managed and controlled optimally. To get results that meet expectations, organizations has adopted approaches that refer to implementation of project portfolio management maturity models. These models aim to maximize the benefits of results of projects and the allocation of resources by the organization. Amid this context, the Project Management Office appear - EGP, protagonists in the process. This article aims to present the results of the application of the maturity model called PMO Maturity Cube to identify the organization's level of grip in the care of the expected results of the process of F Level of Project Portfolio Management of the MR-MPS-SW requirements.*

Keywords: MR-MPS-SW, *PMO Maturity Cube*, *Project Portfolio Management.*

1 Introdução

Com os novos cenários econômicos do mercado, empresas começaram a investir em boas práticas para o gerenciamento de recursos organizacionais e como resultado obtido, evidências para justificativas de investimentos para iniciar ou continuar a execução de um projeto de software (SOFTEX,2016). Uma dessas práticas que vem ganhando notoriedade é a utilização de modelos de maturidade para gerenciamento e monitoramento de portfólio de projetos.

O modelo MR-MPS-SW pertencente ao programa Melhoria de Processos de Software Brasileiro - MPS.Br, possui um processo no nível F que propõe através da implementação do processo de gerenciamento de portfolio de projetos, a qualificação continua para justificar a continuidade ou redirecionamento dos investimentos na execução dos projetos (SOFTEX,2016).

De acordo com um estudo realizado por Rocha et al.,(2015) na implementação do modelo MR-MPS-SW, foram identificadas grandes dificuldades que correspondem a um baixo nível de atuação e maturidade organizacional da empresa nas

[1] PPGIA – Programa de Pós-Graduação de Informática Aplicada Universidade de Fortaleza – UNIFOR - Fortaleza - Ceará {fabioh.sousa@hotmail.com}
[2]PPGIA – Programa de Pós-Graduação de Informática Aplicada Universidade de Fortaleza – UNIFOR - Fortaleza - Ceará {adrianoba@unifor.br}

ações relacionadas ao atendimento da aderência dos processos de gerenciamento de projetos e de portfólio de projetos. Dentre as dificuldades identificadas, foi possível perceber:

- Percebeu-se que mudanças na cultura organizacional que trabalham de uma forma *ad hoc* são difíceis de serem aceitas pelos profissionais da área de projetos e de portfólio de projetos de software.
- A falta de uma equipe que conseguisse passar para os membros da equipe do projeto metodologias, ferramentas, boas práticas, padrões, etc...
- A falta de comprometimento da alta gerência com a implementação dos processos e o fraco apoio da alta direção, pois não existe um alinhamento forte entre os departamentos, principalmente o Escritório de Projetos.
- Dificuldade em relação à estrutura da empresa, por exemplo, a rigidez da estrutura hierárquica da organização, alta rotatividade de pessoal em cargos chaves para a implantação de processos e a empresa sem estabilidade financeira.

No meio de todo esse contexto, um dos protagonistas destas ações de apoio à obtenção da aderência para o modelo MR-MPS-SW é o Escritório de Gerenciamento de Projetos – EGP.

Segundo Hobbs e Aubry (2007, p 74), o EGP é uma unidade responsável por atividades referentes ao gerenciamento de projetos e portfólio de projetos, algumas prosperam e amadureceram, outras perdem forças e apoio, sofrendo cortes e reduções, perdendo suas prioridades, em alguns casos, até mesmo sendo eliminadas. Para identificação dos principais serviços de execução e evolução de maturidade do EGP na organização, foi desenvolvido um modelo de maturidade chamado *PMO Maturity Cube*(PINTO et al.,2010).

Este artigo tem como objetivo apresentar os resultados gerados a partir da aplicação *PMO Maturity Cube* (PINTO et al.,2010), na avaliação de aderência do EGP na implementação do processo de Gerenciamento de Portfólio de Projetos do MR-MPS-SW. Através do mapeamento dos resultados esperados do MPS.br com os serviços propostos pelo *PMO Maturity Cube*, foi possível identificar o quanto o EGP está maduro na suas práticas de portfólio de projetos além da aderência das exigências dos processos de Gestão de portfólio de projetos do nível F do MR-MPS-SW.

Por este artigo tratar-se de uma extensão da dissertação do mesmo autor, intitulada como "Uma abordagem de diagnóstico de Escritório de Projetos de TI baseada no *PMO Maturity Cube*", os modelos selecionados para o estudo de caso no artigo foram os mesmos da dissertação, *PMO Maturity Cube* na aderência do MR-MPS-SW.

Na seção 2, será descrito uma breve definição sobre Escritórios de Gerenciamento de Projetos – EGP. Na seção 3, serão apresentados os modelos de maturidade estudados no artigo, como o Organization Project Management Maturity Model OPM3, o Modelo de Maturidade de Gerenciamento de Projetos – MMGP, o Modelo Project Management Maturity Model – PMMM, o *PMO Mautrity Cube* e o MR-MPS -SW. Na seção 4 será apresentado o mapeamento entre os dois modelos e os resultados da aderência do EGP ao MR-MPS-SW. E na seção 5 são apresentadas as considerações finais e os trabalhos futuros.

2 Escritórios de Gerenciamento de Projetos - EGP

Projetos são vistos como atividades realizadas por um período definido, tendo inicio e fim, concentradas em resultados que tragam valor para a organização. Devem sempre estar alinhadas aos objetivos estratégicos e utiliza-se de recursos humanos, materiais e financeiros restritos (MAXIMILIANO, 2002). Segundo Kerzner(2006), projeto é um empreendimento com um objetivo bem definido, que requer um consumo de recursos , planejamento e opera sob pressão de custos, prazos e qualidade.

Com o grande número de organizações utilizando o gerenciamento de projetos e a disseminação de processos e procedimentos para projetos, tornou-se necessário concentrar essas atividades em uma estrutura que fique responsável pela definição de padrões, reunir e avaliar os projetos, consolidar e monitorar os resultados.

Os Escritórios de Gerenciamento de Projetos - EGPs são unidades organizações que centralizam e coordenam o gerenciamento de projetos e de portfolio sob o domínio de técnicas e ferramentas da disciplina de gerenciamento de projetos, programas, portfolio ou combinação dos três PMI (2004).

Kerzner(2006) ainda diz que, o escritório de gerenciamento de projetos é um centro da corporação para o controle da propriedade intelectual em Gestão de Projetos e ainda tem a responsabilidade de ativamente sustentar o planejamento estratégico da corporação. Os EGPs tornaram-se peças chaves na orientação e apoio dos gerentes no gerenciamento de

projetos, programas e portfólio de projetos, a cobrança passou a ser inevitável na geração de resultados para a organização (PINTO et al.,2010).

3 Modelos de Maturidade

Para a realização deste trabalho foram analisados quatro modelos de maturidade no apoio da aderência do processo de gerenciamento de portfólio de projetos do MR-MPS-SW: *Organizational Project Management Maturity Model - OPM3*, Modelo de Maturidade em Gerenciamento de Projetos - Prado-MMGP, *Project Management Maturity Model - PMMM* e o *PMO Maturity Cube.*

Dos quatro modelos, o *PMO Maturity Cube* foi o modelo que se destacou devido a sua estrutura abordar de forma mais estratégica e funcional as camadas de atuação do EGP: Estratégica, Tática e Operacional, além de mapear os tipos de EGPs nas organizações: Corporativo, Departamental e Programa – Projeto.

Nesta seção, são apresentados de forma detalhada os modelos de maturidades em estudo.

3.1 Organization Project Management Maturity Model - OPM3

O Organization Project Management Maturity Model (OPM3) é um modelo de maturidade de projetos criado pelo PMI. Em 1998, gerentes de projetos pertencentes ao PMI se reuniram e se comprometeram a criar um projeto que tivesse como resultado um modelo padrão que ajudasse as organizações a melhorarem suas capacidades e competências para administrar projetos. Este modelo tem como missão avaliar o nível de maturidade da gerência de projetos, orientar e dar suporte as organizações no que diz respeito ao nível de maturidade no gerenciamento de projetos.

Segundo PMI (2004), o modelo OPM3 é um modelo de maturidade oferecido como base de estudos e auto-avaliação, permitindo com que cada organização possa tomar suas próprias decisões relacionadas a mudanças.

O OPM3 é composto pode três elementos chaves: Conhecimento , Avaliação e Aperfeiçoamento.

O elemento Conhecimento (Knowledge) apresenta todo o conteúdo do modelo. Ele representa um livro (Knowledge Foundation) que contém informações fundamentais sobre o modelo, incluindo aproximadamente 600 melhores práticas em gerenciamento organizacional de projetos e suas expectativas.

No elemento Avaliação (Assessment) são apresentados os métodos, processos e procedimentos para auto-avaliação da maturidade da organização. É aplicado um questionário para identificar as fraquezas e forças da organização comparando com melhores práticas sugeridas. Em alguns casos, a avaliação é realizada por umas ferramentas especificas que trabalham com o modelo.

O elemento Aperfeiçoamento (Improvement) ele proporciona a organização um processo de incremento permitindo com que a mesma possa evoluir o seu nível de maturidade atual para um nível desejado. Segundo OPM3(2003), este elemento é o que tem de melhor comparando-o com outros modelos de maturidade. Na figura 1 pode ser observada a integração desses três elementos.

Figura 1 - Conhecimento, Avaliação e Aperfeiçoamento. Elementos do OPM3. (OPM3,2003)

No modelo OPM3, o nível de maturidade organizacional pode ser avaliado em diversas dimensões:

- ✓ Estágios de melhoria de processos (Standardize, Measure, Control , Improve - SMCI).
- ✓ Domínio de Gerenciamento de Projetos, Programas e Portfólio (Project, Programe Portfólio - PPP
- ✓ Grupos de processo de gerenciamento de projetos: iniciação, planejamento, execução,monitoramento e controle e encerramento (Initiating, Planning, Executing, Controling e Closing – IPECC).

Segundo OPM3(2003), ele sugere que sejam realizados determinador passos para aplicação do modelo:

Passo 1 - Preparação para ser avaliado o nível de maturidade: entendimento dos componentes e objetivos do modelo;

Passo 2 - Avaliação: utilizar a auto-avaliação em um nível de execução para identificar as melhores práticas existentes e necessárias e a localização da organização dentro de amadurecimento continuo. Identificar as capacidades para determinar um nível detalhado e considerar os planos de melhorias. Os resultados sendo satisfatório, a organização poderá finalizar os processos, mas periodicamente avaliar e monitorar as ações em caso de mudanças;

Passo 3 - Plano de Implementação: utilização dos resultados da avaliação e das prioridades definidas pela organização para determinar o escopo e a sequências dos esforços;

Passo 4 - Implementar as Melhorias: Executar as melhorias identificadas e planejadas para prosseguir no melhoramento continuo;

Passo5 - Repetir o processo: Orienta-se executar novamente o Passo 2 para detectar mudanças originadas pelos últimos eventos. Outra sugestão é ir para o Passo 3 para trabalhar com as outras Melhores Práticas identificadas na primeira avaliação.

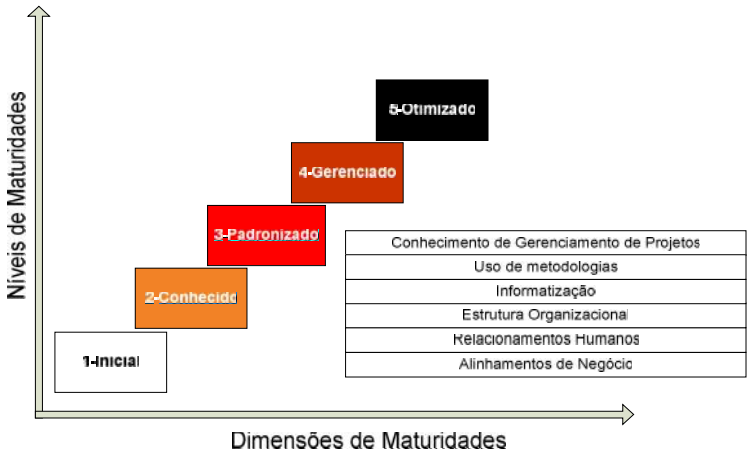
Após a execução dos passos descritos, é possível identificar o nível do modelo de gerência de projetos e da maturidade organizacional da empresa. A partir deste ponto poderá ser reconhecida as potencialidades e resultados que desejam ser atingidos pela organização.

3.2 Modelo de Maturidade de Gerenciamento de Projetos - MMGP

O Modelo de Maturidade de Gerenciamento de Projetos - MMGP , foi desenvolvido por Darci Prado, inicialmente para atender suas necessidades profissionais. Com o passar do tempo, ele foi melhorado e adaptado para ser utilizado pelas organizações na avaliação do nível de maturidade em gerenciamento de projetos e a maturidade organizacional. Este modelo foi lançado para atender duas perspectivas:

O MMGP - Setorial, que foi lançado no final de 2002 e obteve bastante sucesso no Brasil e em alguns países como França e Portugal. O objetivo desta perspectiva é classificar, através de uma avaliação setorial, a habilidade que a organização possui de gerenciar seus projetos em cinco níveis de maturidade: inicial , conhecido, padronizado, gerenciado e otimizado. Após identificação do nível de maturidade, é possível traçar o perfil da organização em seis dimensões: competência técnica, uso prático da metodologia, informatização, estrutura organizacional, alinhamento com os negócios da organização, competências comportamentais e contextuais (PRADO, 2008). Na figura 2 são apresentados os níveis e dimensões de maturidade do MMGP.

Figura 2 - Níveis e Dimensões de Maturidade MMGP - PRADO. (PRADO,2008)



Logo em seguida, foi lançada a perspectiva Corporativa, aplicada em organizações com o objetivo de conhecer estratégias e portfólios. Este modelo tem a organização como um todo, focando na análise ao gerenciamento de portfólio, programas e projetos aos departamentos que estão vinculados fortemente ao planejamento estratégico da organização. O resultado gerado é utilizado como base para uma proposta de maturidade para a organização.

Através de um questionário de 40 questões de múltiplas escolhas, cada nível de 2 a 5 com 10 perguntas cada, é possível obter dados o suficiente para analisar o nível de maturidade que se encontrará a organização. Os resultados são baseados nos seguintes níveis adotados pelo modelo:

No Nível Um Inicial ou Embrionário, apresenta uma desorganização e um desalinhamento entre os envolvidos que trabalham com projetos e as práticas de gerenciamento de projetos. Segundo Prado(2008), os projetos são executados na base da intuição, atos heroicos, boa vontade individual, poucas atividades tem sucesso.

No Nível dois apresenta uma evolução do nível um. Esta evolução se dá principalmente pelo empenho da organização em iniciar boas práticas de gerenciamento de projetos. Neste momento, são realizados investimentos em treinamentos e aquisição de softwares de gerenciamento de projetos. Todo o recurso financeiro utilizado nas aquisições já faz parte do orçamento da empresa.

No Nível três, são percebidos ações de maturidade nas atividades de gerenciamento de projetos. Prado(2008), identifica neste nível, um modelo baseado em uma metodologia, bem desenvolvido, utilizando profissionais capacitadas para cada atividade; uma estrutura organizacional adaptada para o uso de boas práticas; a conciliação do alinhamento estratégico com os negócios da organização; a evolução das "hardskills" e "softskills" dos profissionais envolvidos nos projetos.

Ainda neste nível, as ações de gerenciamento de projetos são centralizadas, fazendo com que surja a necessidade de um departamento responsável pelas mesmas, um escritório de projetos.

No Nível quatro, as competências técnicas já estão definidas como padrão da organização. As boas práticas em gerenciamento de projetos já fazem parte das atividades rotineiras do departamento, além de estarem alinhadas com o negócio da organização. Segundo Prado(2008), ainda neste nível, a prática de melhoria é intensificada, são realizados treinamentos para aprimorar os "hardskills" e "softskills" dos profissionais, são realizadas ações para estimular um relacionamento mais eficaz entre as áreas envolvidas, os gerentes de projetos começam a ter uma participação mais ativa nas decisões da organização.

E finalmente, no Nível cinco, o nível máximo do modelo, tem como objetivo fazer certo as coisas certas sempre levando em consideração a eficiência, eficácia e efetividade. Segundo Prado(2008), este nível representa o estado da arte no Gerenciamento de Projetos em todos os aspectos, deste o primeiro nível. É realizada uma otimização na execução de projetos com base em uma larga experiência, habilidades e conhecimentos dos profissionais envolvidos. Desta forma, algumas áreas como custo, prazo e qualidade são otimizadas. Já existe um forte alinhamento com os negócios da organização e os índices de resultados com sucesso são altos.

Além dos níveis de maturidades descritos anteriormente, o modelo MMGP - PRADO, permite a possibilidade de traçar o perfil da organização em seis dimensões (PRADO,2008):

- ✓ A competência técnica que envolve o nível de maturidade que possuem em relação a conhecimento e práticas em gerenciamento de projetos e outras práticas de gestão.

- ✓ Uso prático de metodologias, como exemplo o Project Management Body of Knowlegde - PMBOK, que dispõe de uma série de boas práticas para obtenção do sucesso nas atividades de gerenciamento de projetos.
- ✓ Informatização como uma ferramenta de facilitação e otimização de diversos aspectos da metodologia, o controle dos projetos pelos principais envolvidos e a criação de uma base comum de conhecimento.
- ✓ A adequação da estrutura organizacional que tem como propósito maximizar os resultados e mitigar os conflitos.
- ✓ Alinhamento com os negócios da organização que permite um relacionamento harmonioso dos projetos com as estratégias organizacionais.
- ✓ Competências comportamentais e contextuais que visam questões relacionadas as ações humana do profissional e o seu comportamento no ambiente organizacional.

3.3 Modelo de Maturidade de Gerenciamento de Projetos – PMMM

O modelo Project Management Maturity Model - PMMM, foi criado pelo Dr. Harold Kerzner em 2002, uma extensão do modelo CMM. Este modelo visa como objetivo definir o estágio atual, planejar e implementar ações para o desenvolvimento gradual no gerenciamento de Projetos (KERZNER,2006).

O PMMM faz um tipo de combinação dos níveis de maturidade do CMM com a estrutura de áreas de conhecimento do PMBOK. Ele foca praticamente no gerenciamento e processos de mudanças na cultura da organização para a aplicação de boas práticas em gerenciamento de projetos.

Este modelo apresenta cinco níveis de desenvolvimento para a implementação, acompanhamento e alcance da excelência em gerenciamento de projetos (KERZNER,2006):

Nível 1 - Linguagem Comum: Esta é a fase inicial do processo de maturidade. neste nível a organização ainda não apoia o gerente de projetos, mas passa a reconhecer a importância de ter metodologias e boas práticas para o gerenciamento de projetos. Neste momento, a organização inicial os preparos para práticas de desenvolvimento e treinamentos de técnicas, como também a certificação dos gerentes de projetos, disponibilidade de ferramentas e a definição de uma terminologia para o gerenciamento de projetos.

Nível 2 - Processos Comuns: Neste nível, a organização reconhece a necessidade de se criar processos de padrão organizacional. Passa-se a aplicar algumas técnicas de escopo, prazos e custos acompanhados pela figura do gerente de projetos. Todas as ações possuem como subsidio informações históricas de projetos que tiveram sucesso.

Nível 3 - Metodologia Singular: Neste nível, a organização reconhece a possibilidade e necessidade da integração dos processos e metodologias centradas no gerenciamento de projetos.

Nível 4 - Benchmarking: Neste nível, a organização passa a executar práticas de comparação de boas práticas em gerenciamento de projetos realizadas por outras organizações. Neste momento, a organização tenta identificar as melhores práticas e definir como aplicá-las para a melhoria de suas atividades.

Nível 5 - Melhoria Continua: Neste nível, a organização já se apresenta com mais maturidade, utilizando informações obtidas nos níveis anteriores para praticar ações de melhorias continua em gerenciamento de projetos da organização.

Os níveis de maturidade do modelo PMMM de Kerzner(2006)podem ser avaliados de acordo com o ciclo de vida da organização para o alcance da maturidade desejada no gerenciamento de projetos. Esta avaliação divide-se em cinco fases:

Fase embrionária: Os gerente de projetos seniores conhecem que existe a grande possibilidade de o gerenciamento de projetos resolver problemas relacionados a organização a partir da base.

Fase de aceitação pela gerência executiva: A falta de apoio da gerência é a maior dificuldade do alcance do nível de maturidade e excelência. Neste momento, é preciso tornar visível todo o processo diante a gerência para ganhar o seu apoio.

Fase de apoio dos gerentes da área: Nesta fase, os gerentes se comprometem com a gestão de projetos, eles passam a conhecer as ferramentas e práticas de gerenciamento de projetos e passam a dar um apoio ostensivo e comprometido aos projetos.

Fase de crescimento: Neste momento podem ocorrer as outras fases em paralelo. A organização já compreende a grande importância do ciclo de vida do projeto no gerenciamento de projetos. São definidas metodologias comprometidas com

o planejamento, mitigando as oscilações de mudanças em relação ao escopo, prazos e custos. Tudo passa a ser monitorado por um sistema de rastreamento para manter o controle.

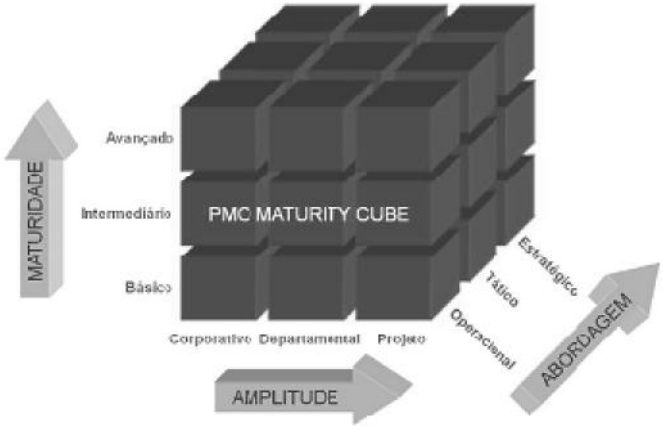
Fase de maturidade: Nesta fase, o foco passa a ser o acompanhamento e controle dos custos e prazos, são utilizados sistemas de contabilidade horizontal, incluindo-se indicadores de valor agregado que definam com precisão as verbas gastas em cada uma das atividades do projeto, além de identificar estimativas precisas, prazos tangíveis e custos realistas. No final desta fase é gerado um sistema de ensino continuo de longo prazo baseado em lições aprendidas.

3.4 *PMO Maturity Cube*

O *PMO Maturity Cube* (PINTO et al.,2010) é um modelo de avaliação de maturidade de Escritórios de Projetos em Gerenciamento de Projetos. Ele reuni conceitos referentes a amplitude do escritório que são válidos para corporativos , departamentais e estratégicos. Conceitos relacionados aos serviços prestados como: estratégicos, táticos e operacionais. E ainda apresenta através do preenchimento de um questionário, o nível de maturidade que encontra-se o EGP, que pode ser básico, intermediário e avançado.

As três dimensões mencionadas no parágrafo anterior, formam um cubo que trabalha através de um questionário, pode ser aplicado nos EGPs para identificar o nível de maturidade. Na Figura 3 é possível ver como essas dimensões estão dispostas.

Figura 3 – As três dimensões do PMO Maturity Cube. (PINTO et al.,2010)

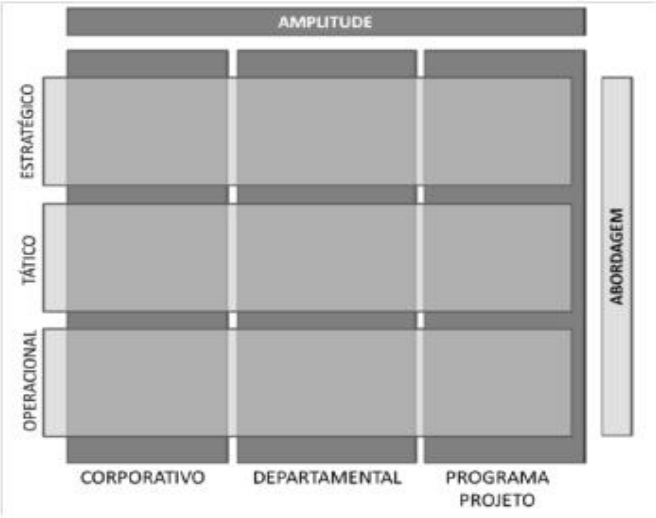


Os EGPs com suas atribuições, permitem ser classificados por suas amplitudes, que são funções executadas na organização: Corporativo, que abrange a organização, Departamental, que abrange uma área, departamento, diretoria ou unidade de negócio, e o Programa - Projeto, que tem como abordagem apenas um projeto ou programa da organização. Existem também as abordagens que os EGPs atuam que são: estratégicos, táticos e operacionais.

O EGP estratégico oferece aos seus clientes serviços que estejam de alguma forma ligados ao planejamento estratégico da organização, tais como gerir o portfólio, monitorar resultados de objetivos estratégicos, prover informações a alta gestão para tomadas de decisão, priorizar projetos , monitorar e realizar estratégias, entre outros (PINTO et al.,2010).

O EGP tático tem como objetivo oferecer a seus clientes serviços direcionados a grupos de projetos ou indivíduos, tais como prover metodologias de gerenciamento de projetos, fornecer ferramentas, aplicações e treinamentos para os gerentes de projetos e as equipes, entre outros (PINTO et al.,2010). O EGP operacional tem como objetivo apoiar o planejamento e controle do projeto, fazer *coaching/mentoring*, gerenciar um projeto estratégico e recuperar projetos com problemas, entre outros (PINTO et al.,2010). Na figura 4 é possível verificar os tipos de amplitude e abordagens atuantes nos EGPs.

Figura 4 - Amplitude e Abordagens dos Escritórios de Projetos (PINTO et al.,2010)



Para cada EGP, existe um questionário de acordo com a sua amplitude, o modelo identifica quais serviços são oferecidos ao EGP sob cada abordagem e identifica o nível de sofisticação da execução de cada atividade.

Os questionários, corporativo, departamental e operacional, são divididos em três partes, cada parte avalia os serviços oferecidos pelo EGP, serviços tais como: Estratégicos, Táticos e Operacionais. Na pergunta, o EGP informa o nível de maturidade atual de seus serviços oferecidos naquela amplitude e informa o nível que deseja alcançar. As informações contidas nos questionários foram selecionadas a partir de uma pesquisa publicada pelo Dr. Hobbs e a Dra. Aubry. Esta pesquisa identificou 27 funções mais comuns executadas pelos EGPs. Elas representam os serviços mais prestados que tornaram-se referências para o modelo(HOBBS E AUBRY,2007).

Após a coleta de dados pelo questionário, o nível de maturidade atual e o desejado serão dados em percentuais, representando o nível básico o percentual que vai de 0% à 33%, o nível intermediário que vai de 34% à 60% e o nível avançado que vai de 61% à 100%. Na tabela 1 é apresentado o resultado da coleta de dados utilizando o *PMO Maturity Cube*.

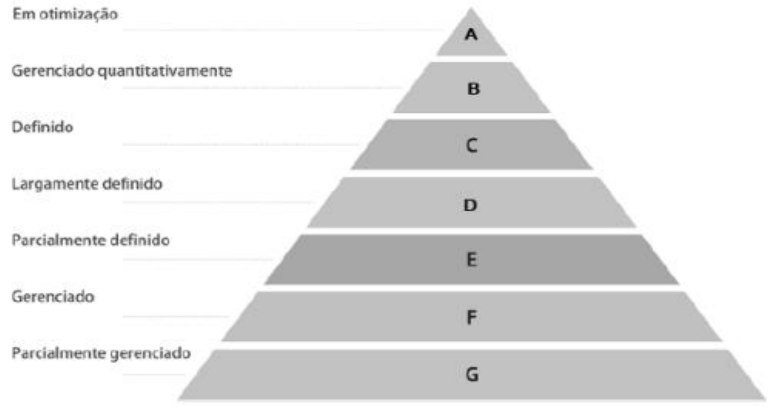
Tabela 1-Resultados dos dados coletados em cima do modelo PMO Maturity Cube (PINTO et al.,2010).

Empresa	Setor	Amplitude do PMO			Abordagem do PMO		
					Avaliação Estratégica	Avaliação Tática	Avaliação Operacional
A	Bens de Consumo	Corporativa	Maturidade	Nível Atual	33% Básico	41% Intermediário	57% Intermediário
				Nível Desejado	53% Intermediário	62% Intermediário	65% Intermediário

3.5 MR-MPS-SW Nível F – Processo Gestão de Portfólio de Projetos

O Programa Melhoria de Processo de Software Brasileiro foi criado pela Softex com o apoio do Ministério da Ciência e Tecnologia (MCT), Financiadora de Estudos e Projetos (FINEP), Serviço Brasileiro de Apoio às Micro e Pequenas Empresas (SEBRAE) e do Banco Interamericano de Desenvolvimento (BID). O objetivo do MPS.Br é ser um modelo de maturidade de processos de software pra que empresas atinjam mais rápido o nível de maturidade. O MPS.Br possui sete níveis de maturação que trabalham a implantação gradual do modelo de maturidade de processo na organização. Na Figura 5 são apresentados os setes níveis.

Figura 5 – Níveis de maturidade do MPS.Br. (SOFTEX,2016).



O processo de Gerência de Portfólio de Projetos - GPP tem como objetivo apoiar o inicio e manter os projetos que sejam necessários, suficientes e sustentáveis, de forma a atender os objetivos estratégicos da organização (SOFTEX,2016).

O processo de gerência de portfólio de projetos do modelo MR-MPS-SW tem como foco comprometer os investimentos e recursos organizacionais adequados e estabelece a autoridade necessária para executar os projetos selecionados. Ele executa a qualificação continua de projetos para confirmar que eles justificam a continuidade dos investimentos, ou podem ser redirecionados para justificar (SOFTEX,2016).

Comparando ao processo de Gerenciamento de Projetos do Nível G, enquanto este processo está focado em envolver atividades para gerenciar os projetos, o processo Gerência de Portfólio de Projetos está focado em atividades relacionadas ao gerenciamento do conjunto de projetos da carteira da organização, atividades como seleção, estudo de viabilidade e valor agregado aos objetivos estratégicos da organização.

4 Avaliação da maturidade da gestão de portfólio de projetos

4.1 Mapeamento dos modelos e experiência de uso

Para trabalhar de forma consistente o modelo *PMO Maturity Cube* foi criado um mapeamento vinculando os resultados esperados do processo do Gerenciamento de Portfólio de Projetos do MR-MPS-SW com os serviços corporativos apontados pelo *PMO Maturity Cube*. Na tabela 2 é apresentado o mapeamento entre os dois modelos.

Para a elaboração do mapeamento foram analisados os escopos de cada nível de maturidade dos serviços do *PMO Maturity Cube* com as necessidades de atendimento de cada resultado esperado do processo GPP. Assim, de acordo com o nível de maturidade do serviço obtido pelo EGP é possível comprovar o quanto ele está preparado para atender as exigências do processo em questão.

Tabela 2 – Mapeamento dos resultados esperados com os serviços do PMO Maturity Cube.

Resultados Esperados GPP	Serviços do *PMO Maturity Cube*	Nível Maturidade
Serviços e resultados esperados aplicados na abordagem estratégica		
GPP1 – Identificação de oportunidades e necessidades	O PMO identifica, seleciona e prioriza novos projetos	Nível 3 - O PMO estabelece para toda a organização um processo formal de identificação, seleção e priorização de novos projetos baseado em categorias e critérios pré-estabelecidos.
GPP4 – Monitoramento do	O PMO gerencia um ou	Nível 2 - O PMO possui uma lista dos

portfólio	mais portfólios	projetos ativos de toda a organização e busca a sua priorização, porém de forma não estruturada.
GPP5 – Ações de correção do portfólio	O PMO gerencia um ou mais portfólios	Nível 3 - O PMO possui uma lista dos projetos ativos e priorizados de toda a organização e estabelece processos formais, atuando como facilitador na definição (identificação, categorização, avaliação, seleção), desenvolvimento (priorização, balanceamento e autorização) e execução (monitoramento, revisão e gestão de mudanças) do portfólio.
GPP7 – Estudo de viabilidade dos projetos	O PMO gerencia os benefícios de projetos ou programas	Nível 1 - O PMO acompanha a evolução da realização dos benefícios esperados para a organização, apenas durante a realização do mesmo, avaliando os resultados de negócio e comparando-os com os objetivos estratégicos originais da organização vinculados ao planejamento estratégico.
Serviços e resultados esperados aplicados na abordagem tática		
GPP2 – Identificação de recursos e orçamento para o projeto	O PMO atua na alocação e movimentação de recursos entre os projetos	Nível 1 - O PMO atua de forma reativa na alocação/movimentação de recursos entre projetos de toda organização.
GPP3 – Estabelecida a autoridade e responsabilidade pelo projeto	O PMO desenvolve e implementa a metodologia padrão de Gerenciamento de Projetos	Nível 3 - O PMO desenvolveu a metodologia padrão para a organização e esta é utilizada por todos os projetos e de forma correta.
GPP6 – Tratamento dos conflitos de recursos	O PMO atua na alocação e movimentação de recursos entre os projetos	Nível 3 - O PMO possui uma visão do pool de recursos e autoridade para alocar e mover recursos entre projetos de toda organização.
GPP8 – Comunicação com as partes interessadas sobre o portfólio de projetos	O PMO gerencia interfaces com clientes	Nível 2 - O PMO gerencia o relacionamento com clientes dos projetos da organização, administrando expectativas e avaliando a satisfação, porém sem autoridade para influenciar diretamente a gestão dos projetos envolvidos.

4.2 Coleta e análise dos dados

Para a realização desta pesquisa, foi selecionada uma empresa privada nacional localizada em Fortaleza-Ceará, que trabalha com projetos de desenvolvimento de software e possui escritório de gerenciamento de projetos. O EGP trabalha vinculado a área de gerência de TI que possui profissionais com um bom nível de maturidade de gerenciamento de projetos de softwares.

Na pesquisa foi aplicado, através do envio de um arquivo digital por email ao líder do EGP, um questionário(PMO.M.C,2010) do modelo *PMO Maturity Cube* no EGP para coletar os dados. Este questionário era formado por 27 questões que monitoram as abordagens estratégicas, táticas e operacionais do EGP.

Após a coleta e consolidação dos dados pelo questionário, foi possível identificar a capacidade do EGP para atender as exigências dos resultados esperados.

De acordo com a tabela 3, pode ser visto como o mapeamento entre o resultado esperado, o serviço do EGP pelo *PMO Maturity Cube*, o nível de capacidade do serviço obtido pelo EGP e o nível exigido para atendimento da aderência do MR-MPS-SW.

Na coluna referente ao Nível *PMO Maturity Cube* são apresentados os níveis de maturidade do EGP entrevistado naquele serviço especifico, na coluna Nível do MR-MPS-SW são apresentados os níveis de maturidade para atendimento dos resultados esperados exigidos pela necessidade da aderência do MR-MPS-SW. Desta forma, para que o EGP possua maturidade o suficiente para atendimento do resultado esperado, o nível da coluna Nível *PMO Maturity Cube* deve ser igual ou superior ao nível da coluna Nível MR-MPS-SW.

Tabela 3 – Dados coletados, analisados e comparados.

SIGLA	Resultado Esperado	Serviço do EGP	Nível PMO Maturity Cube	Nível MR-MPS-SW
Serviços estratégicos				
GPP1	Identificação de oportunidades e necessidades	3. O PMO identifica, seleciona e prioriza novos projetos.	Nível 3	Nível 3
GPP4	Monitoramento do portfólio	2. O PMO gerencia um ou mais portfólios	Nível 2	Nível 2
GPP5	Ações de correção do portfólio	2. O PMO gerencia um ou mais portfólios	Nível 2	Nível 3
GPP7	Estudo de viabilidade dos projetos	4. O PMO gerencia os benefícios de projetos ou programas	Nível 1	Nível 1
Serviços Táticos				
GPP2	Identificação de recursos e orçamento para o projeto	9. O PMO atua na alocação e movimentação de recursos entre os projetos	Nível 1	Nível 1
GPP3	Estabelecida a autoridade e responsabilidade pelo projeto	6. O PMO desenvolve e implementa a metodologia padrão de Gerenciamento de Projetos	Nível 4	Nível 3
GPP6	Tratamento dos conflitos de recursos	9. O PMO atua na alocação e movimentação de recursos entre os projetos	Nível 1	Nível 3
GPP8	Comunicação com as partes interessadas sobre o portfólio de projetos	8. O PMO gerencia interfaces com clientes	Nível 1	Nível 2

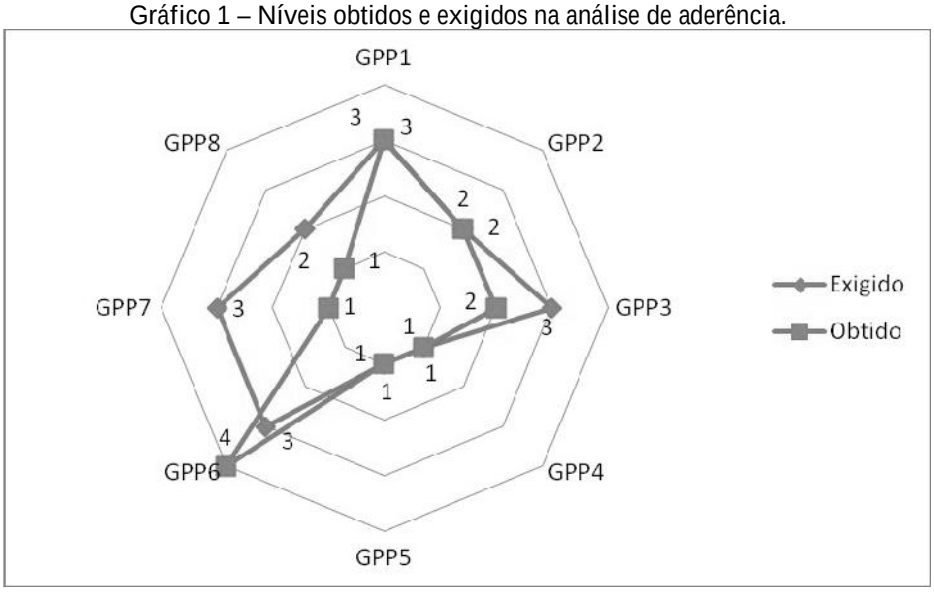
Para o processo de gerenciamento de portfólio, os resultados esperados foram mapeados apenas para nas abordagens estratégica e tática, para o operacional não foram apresentadas características de atuação dos serviços nos resultados esperados.

Na abordagem estratégica, o EGP conseguiu atingir os resultados esperados GPP4 e GPP7, já os GPP1 e GPP5 , o EGP não apresentou maturidade o suficiente para atender as necessidades dos resultados esperados.

Na abordagem tática, o EGP conseguiu atingir os níveis de maturidades para atendimento dos resultados esperados GPP2 e GPP3, para os resultados GPP6 e GPP8, o EGP ficou abaixo do esperado para atender as exigências do atendimento dos resultados.

No gráfico 1 é possível observar os níveis de maturidade do EGP alcançados em comparação aos níveis exigidos para aderência do MR-MPS-SW.

Gráfico 1 – Níveis obtidos e exigidos na análise de aderência.



Na tabela 4 é apresentado o resultado do nível de maturidade do EGP nas abordagens estratégica, tática e operacional.

Tabela 4 – Análise do resultado da aplicação do PMO Maturity Cube.

Empresa	Setor	Amplitute			Abordagem do PMO		
					Avaliação Estratégica	Avaliação Tática	Avaliação Operacional
A	Prestação de serviços de gerenciamento de projetos de software	Corporativo	Maturidade	Nivel Atual	29,03% Básico	29,03% Básico	00,00% Intermediário
				Nível Desejado	46,77% Intermediário	43,55% Intermediário	00,00% Intermediário

De acordo com a análise dos resultados dos niveis de maturidadae do EGP nas abordagens estratégica, tática e opercional e nos atendimentos dos resultados esperados do processo de gerenciamento de portfolio de projetos , o EGP trabalha tanto de uma forma estratégica com tática nos projetos, na sua evolução de maturidade desejada , a atuação estratégica sobrepoe a tática. Já na avaliação operacional, o modelo *PMO Maturity Cube* não possui serviços que mapeados consigam atender esta abordagem.

5 Considerações finais

Neste trabalho foi possível constatar a eficiência e abrangência da aplicação do *PMO Maturity Cube* como apoio a análise de aderência na implementação do processo de gerenciamento de portfólio de projetos do MR-MPS-SW. Com o mapeamento entre os dois modelos foi possível realizar um diagnóstico do EGP e identificar também o nível de maturidade organizacional.

De acordo com os resultados gerados pelo estudo observou-se que o EGP em questão não está preparado para atender satisfatoriamente as exigências para implementação do processo de Gerenciamento de Portfólio de Projetos do Nível F do MR-MPS.Br. Na avaliação da maturidade organizacional, o EGP se saiu bem no atendimento dos serviços estratégicos e táticos.

Como limitação deste trabalho, o referencial teórico sobre o modelo PMO Maturity Cube é restrito, devido o modelo ser recente no mercado e possuir poucas fontes de leitura.

Para trabalhos futuros, será desenvolvida uma abordagem dinâmica para a utilização do *PMO Maturity Cube* no apoio à análise de aderência do processo de Gerenciamento de Projetos do Nível G do MR-MPS-SW.

Agradecimentos

Agradeço, em particular, o apoio irrestrito da Faculdade Metropolitana da Grande Fortaleza – Fametro na realização deste trabalho acadêmico.

Referências

HOBBS, B.; & AUBRY, M. A Multi-Phase Research Program Investigating Project Management Offices (PMOs): The Results of Phase 1. Project Management Journal, 38, 74-86, 2007.

KERZNER, Harold. *Gestão de Projetos: as melhores práticas.* Tradução: Lene Belon Ribeiro. 2. ed.São Paulo: Bookman, 2006.

MAXIMILIANO, A. C. A.. Administração de projetos: como transformar ideias em resultados. 2ª ed. São Paulo: Atlas, 2002.

OPM3, Project Management Institute. Organizational Project Management Maturity Model (OPM3): *Knowledge Foundation*. Newton Square, Pennsylvania: Project Management Institute Inc., 2003.

PINTO, A.; COTA, M.; LEVIN, G. Paper: "*PMO Maturity Cube*", um modelo de avaliação de maturidade exclusivo para Escritórios de Projetos.Washington DC, 11 a 13 de julho, 2010.

PMO.M.C(2010), Paper:"*PMO Maturity Cube*", um modelo de avaliação de maturidade exclusivo para Escritórios de Projetos. In: "https://issuu.com/americopinto/docs/pmomaturitycubepor ", último acesso em 30/03/2016

PMI, The Standard for Portfolio Management, 2006. - Third Edition. Newtown Square, PA: Project Management Institute Inc., 2004.

PRADO, D. S. do. Maturidade em Gerenciamento de Projetos. Nova Lima: INDG Tecnologia e Serviços Ltda, 2008. 7 v. (Série Gerenciamento de projetos).

ROCHA A. R.; Dificuldades e Fatores de Sucesso na Implementação de Processos de Software Utilizando o MR-MPS e o CMMI, Universidade Católica de Brasília, 2015

SOFTEX (2016), "MPS.BR: Melhoria de Processo do Software Brasileiro". In: http://www.softex.br/mpsbr/, último acesso em 05/05/2016.

Um estudo exploratório sobre o uso de algoritmos genéticos para o problema de eficiência energética em trens urbanos

Mayrton Dias de Queiroz [1]
Marcelle Batista Martins [1]
Rodrigo Gonçalves Daniel [1]
Natasha Correia Queiroz Lino [1]

Resumo: *Devido ao aumento populacional, podem-se observar alguns problemas da atualidade, tais como, o desperdício de recursos naturais, congestionamentos, poluição sonora e do ar, dentre outras questões, de forma que se faz necessário a adoção de medidas para minimizar tais problemas. Dentro desse cenário, podem-se destacar as Cidades Inteligentes, que tem como objetivo unir as cidades com as tecnologias de informação retornando assim uma solução inteligente para a sociedade. A Mobilidade Urbana é um dos eixos mais importantes inseridos nas Cidades Inteligentes, qual recebem atenção de vários pesquisadores, onde existe uma preocupação sobre como diminuir a quantidade de veículos nas ruas, reduzir o consumo de combustíveis fósseis, entre outros. Uma alternativa que ajuda a minimizar um dos problemas relatados é a utilização dos transportes urbanos como, por exemplo, os trens elétricos, contudo, surge um problema com relação ao consumo elevado da energia elétrica. Sabendo dessa problemática alguns autores aderem como solução para o problema do consumo energético, a utilização dos Algoritmos Genéticos a fim de encontrar os melhores perfis de condução e velocidade. Assim, este trabalho tem como objetivo fazer uma comparação sobre os possíveis tipos de implementação aplicando os operadores de Crossover encontrados na literatura para que sejam aprimorados possíveis algoritmos evolucionários, de forma a propor novas soluções para reduzir o consumo energético nos trens elétricos.*

***Palavras-chave*:** Cidades Inteligentes. Eficiência Energética. Algoritmos Genético.

Abstract: *Due to population growth, it may be observed some present-day problems such as the waste of natural resources, traffic congestion, noise and air pollution, among other issues, so it is necessary to adopt measures to minimize such problems. In this scenario, one can highlight the Smart Cities, which aims to unite the cities with the information technology thus returning an intelligent solution to society. The Urban Mobility is one of the most important axes entered in Smart Cities, which receive attention of many researchers, where there is a concern about how to reduce the number of vehicles on the streets, reduce the consumption of fossil fuels, among others. An alternative which helps minimize one of the problems reported is the use of urban transport, for example, electric trains, however, a problem arises with respect to the high consumption of electrical energy. Knowing of this problem some authors adhere as a solution to the problem of energy consumption, the use of genetic algorithms in order to find the best driving and speed profiles. This work aims to make a comparison on the possible types of implementation by applying the Crossover operators found in the literature to possibly improve evolutionary algorithms, in order to propose new solutions to reduce energy consumption in electric trains.*

***Keywords*:** Energy Efficiency. Smart Cities. Genetic Algorithms.

1 Introdução

Ao observar o desenvolvimento populacional durante os últimos anos, um dos números que vem chamando a atenção de pesquisadores é com relação ao valor do crescimento populacional. Segundo a Organização das Nações Unidas [1], atualmente a população mundial é composta por 7,3 bilhões de pessoas. No entanto, as previsões

[1]Centro de Informática(CI), UPPB, Campus 1, R. dos Escoteiros - João Pessoa (PB) - Brasil
mayrtondias@ppgi.ufpb.br,marcellebm@gmail.com,rodrigodaniel@cc.ci.ufpb.br,natasha@ci.ufpb.br

presentes no relatório "Perspectivas da População Mundial: A Revisão de 2015", estima-se que a população atingirá a marca de 8,5 bilhões até 2030, e de 9,7 bilhões até 2050. Seguindo esse ritmo em 2100 chegará a 11,2, ou seja, um crescimento de 53% comparado com os dados do ano de 2015.

Com o aumento populacional atual, surgem diversas preocupações como destacado por [2]. Ele defende que um planeta onde a população cada vez mais torna-se urbana, dando origem as megacidades, ou seja, cidades com populações acima de 10 milhões de habitantes, é preciso desenvolver modelos de sustentabilidade urbana capazes de alinhar o desenvolvimento desses espaços com o respeito aos princípios da sustentabilidade.

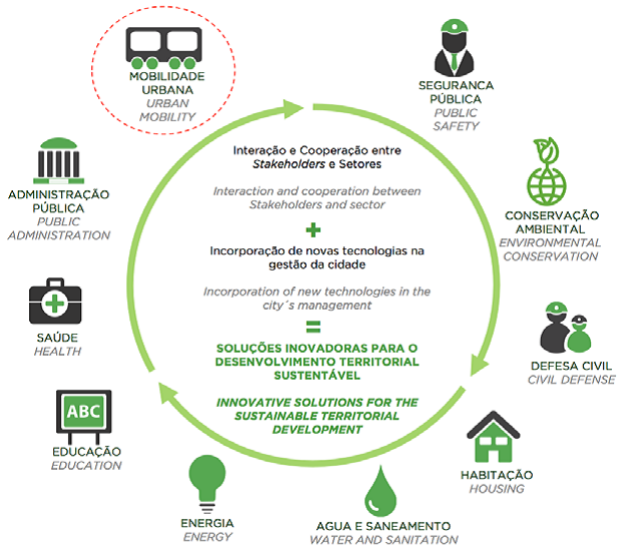
As cidades sustentáveis devem se basear em um modelo no qual consiga equilibrar, de forma eficiente, os recursos necessários para o seu funcionamento, como terra urbana e recursos naturais, água, energia, alimento, entre outros. Encontrar um modelo de funcionamento, gestão e crescimento distintos dos modelos anteriores que se baseava em simplesmente em "expansão com esgotamento" é um dever da cidade sustentável.

Como mostrado em [3], no contexto em que a população mundial se depara com o desafio de alcançar padrões sustentáveis de desenvolvimento já existem várias pesquisas com o intuito de buscar novas soluções para as cidades, visto que, o mundo pede cidades humanas, sustentáveis e inteligentes.

Diante dessas circunstâncias, em [4] é destacado o conceito de Cidade Inteligente (do inglês *Smart City*) que parte da perspectiva na qual a tecnologia é fator indispensável para que as cidades possam se modernizar e oferecer melhor infraestrutura à população. Além disso, esse conceito tem se mostrado fundamental no processo de tornar os centros urbanos mais eficientes e de oferecer boa qualidade de vida e gestão dos recursos naturais por meio de um processo participativo.

Conforme exposto por [4], o autor faz uma divisão dos pontos principais como as atividades de segurança pública, gestão de resíduos e da água, administração pública, saúde, educação, segurança, mobilidade, energia e habitação, entre outros. Onde as cidades devem direcionar-se pelos eixos (mostrado na Figura 1), visto que os mesmos direcionaram as cidades para um melhor desenvolvimento. Esse trabalho tem foco na Mobilidade Urbana, pois como afirma [5] a cidade deve ter um inteligente sistema logístico e de transporte de pessoas, ter meios eficientes de acessibilidade local e internacional, ter um sistema de transporte sustentável - não agressivo ao meio ambiente - e ter amplo acesso à internet.

Figura 1: Mecanismo das cidades inteligentes Fonte: [4]



Em relação aos tipos de transportes urbanos, pode-se destacar os Trens Unidade Elétrica (TUE), como uma forma de minimizar os efeitos da emissão de gases poluentes, diminuição de veículos individuais, bem como também a redução do consumo dos combustíveis fósseis. No entanto, surge um problema com relação ao gasto de energia, fazendo-se necessária a adoção de medidas que visam a eficiência energética.

No trabalho [6] é proposto o uso de Algoritmo Genético (em inglês, Genetic Algorithm GA) para determinar o melhor perfil de condução a ser seguido pelos maquinistas, a fim de encontrar a eficiência energética. A autora sugere um GA que tem objetivo de gerar estratégias de controle de tráfego energeticamente eficientes.

De maneira geral, esse trabalho consiste em fazer um comparativo entre os tipos de operadores genéticos de *crossover* existentes na literatura e que atendam as necessidades para solucionar o problema de eficiência energética em trens urbanos. Para facilitar a visualização do comportamento de cada operador de *crossover* foi desenvolvido um *software* com uma GUI (*Graphic User Interface*).

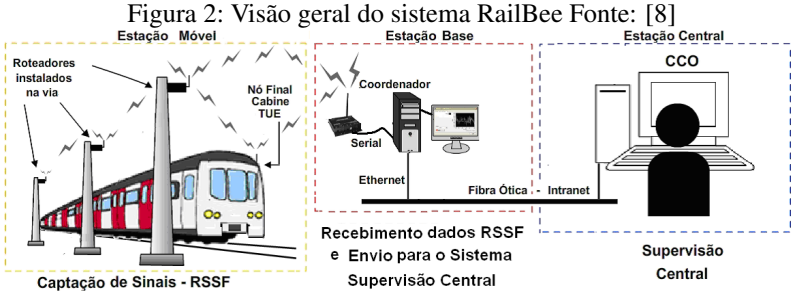
O restante deste trabalho está estruturado da seguinte forma: Na Seção 2 é apresentado detalhes sobre os trens Urbanos de Recife. Na Seção 3 são descritos os pontos relevantes dos GA, bem como também os operadores encontrados na literatura. Na Seção 4, metodologia realizada no trabalho, Na Seção 5 , os dos resultados gerados do software desenvolvido e na Seção 6 as conclusões e trabalhos futuros.

2 Trens Urbanos

Conforme citado anteriormente, um dos eixos das Cidades Inteligentes é o da Mobilidade Urbana. Sendo assim, pode-se notar que na cidade de Recife existe uma alternativa para atender os requisitos da Mobilidade Urbana, que são os trens elétricos. O METROREC é uma empresa operadora de transporte urbano sobre trilhos inserida no Sistema de Transportes Públicos de Passageiros, da Região Metropolitana do Recife [7].

Um dos pontos relevantes do METROREC é com relação as pesquisas desenvolvidas, por exemplo, o sistema Railbee. Segundo [8] Railbee é um sistema de telemetria de sinais de trens desenvolvido e implementado nos TUE da Companhia de Transporte Metroferroviário que é composto por três subsistemas (Figura 2), sendo o primeiro deles um subsistema telemétrico para captação de sinais na via de tráfego, o segundo como um subsistema de recebimento, decodificação e envio dos dados recebidos pelo sistema de captação para o sistema de supervisão e por fim um subsistema de supervisão para armazenamento e visualização dos dados processados pelo sistema.

Figura 2: Visão geral do sistema RailBee Fonte: [8]



Esse sistema, denominado Sistema Telemétrico Dinâmico RailBee, é capaz de obter dados de sensores remotos dentro da cabine do TUE, processá-los e enviá-los à centrais de controle e monitoramento. Ele utiliza redes de sensores sem fio, com tecnologia ZigBee para processamento e distribuição dos dados coletados, possibilitando o monitoramento e a distribuição de informações sobre o desempenho dos trens na via de tráfego [8].

Cada TUE é composto por 4 carros, sendo 2 carros-motor e 2 carros-reboque. Os carros-motor são localizados nas extremidades do TUE, onde estão instalados a cabine de comando e os motores de tração, e sua capacidade de lotação é de 268 passageiros, sendo 61 sentados e 207 em pé. Os carros-reboque são os carros centrais, que possuem uma capacidade de lotação é de 289 passageiros, sendo: 72 sentados e 217 em pé.

3 Algoritmos Genéticos

No ano de 1975, Holland publicou o livro *"Adaptation in Natural and Artificial Systems"*. Em seu trabalho, Holland apresenta os GA como uma metáfora para os processos evolutivos, de forma que ele pudesse estudar a adaptação e a evolução no mundo real, simulando dentro de computadores [9]. GA é um ramo dos algoritmos

evolucionários, e como tal, podem ser definidos como uma técnica de busca baseada numa metáfora do processo biológico de evolução natural [9]. Os GA são técnicas de heurísticas de otimização global.

3.1 Um algoritmo básico

Para solucionar um problema com GA tem-se a necessidade de analisar os passos básicos do algoritmo, a fim de identificar quais as adaptações necessárias para que o mesmo atenda de maneira efetiva ao problema. Dessa forma, pode-se notar o algoritmo básico proposto por [9] segue os seguintes passos:

1. Inicialização da população
2. Avaliação individual de cada cromossomo da população gerada
3. Seleção dos pais da população na geração atual
4. Aplicação dos operadores de crossover e de mutação no pais selecionados anteriormente para geração de novos cromossomos da próxima geração
5. Exclusão dos pais da população atual
6. Avaliação dos filhos e inserção na nova população
7. Verificação se já atende ao número de gerações, ou o melhor individuo já satisfaz os requerimentos e desempenho, dessa forma ele será retornado, caso contrário ira voltar ao passo 3.

3.2 Representação cromossomial

A representação cromossomial é fundamental para o algoritmo genético, sendo a maneira básica de traduzir a informação do nosso problema em uma maneira viável de tratamento pelo computador [9]. Em outras palavras, a representação cromossomial é a forma de conectar as propriedades relevantes do mundo real ao algoritmo genético, com a finalidade de se obter o melhor resultado possível.

Existem vários tipos de representação cromossomial, como a binária, a de números reais, permutação de elementos para GA baseado em ordem, lista de regras, ou qualquer estrutura de dados necessária. Neste trabalho, foi acolhida a representação de números reais, visto que, a precisão é um fator importante para o problema. No caso da representação binária, existe o problema da necessidade de muito bits para ter uma maior precisão, bem como também um maior gasto de tempo na conversão de real para binário e de binário para real novamente.

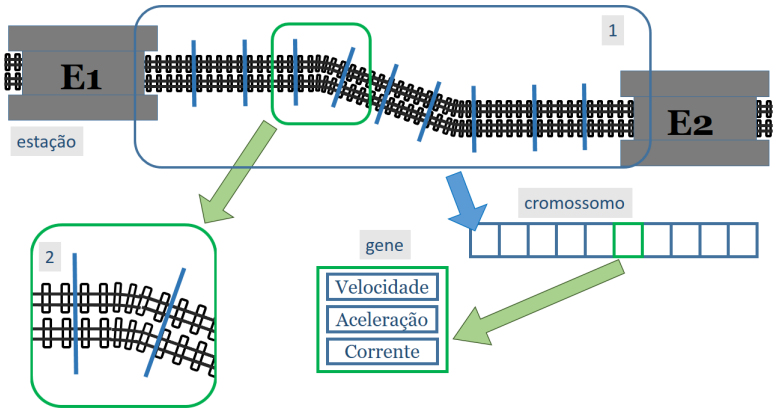
Na Figura 3 observa-se a representação adotada neste trabalho, onde E1 e E2 são as estações de embarque e desembarque dos trens. No ponto 1, está a representação do trilho que possui 10 divisões, com base nas divisões existentes entre duas estações no METROREC. Essas divisões servem para que cada seção não passe corrente para a outra, pois cada valor da corrente representa um valor para a velocidade, a qual auxilia o maquinista no controle da velocidade máxima de cada seção.

Então, com base nessa configuração própria do problema, pode-se tirar proveito dessa situação associando-a ao algoritmo genético, uma vez que cada seção possui um valor de corrente elétrica específica, podendo ser considerado um gene na representação cromossômica, como mostrado pela seta azul na Figura 3. Outro ponto importante é observar que com a ajuda do zigbee são capturados dados como velocidade, aceleração e corrente que podem ser considerados um gene, um vez que esses valores podem ser distintos de uma seção para outra.

3.3 Escolha da população inicial

Ao analisar as formas de criação da população inicial, pode-se citar como exemplo o método de criação aleatória, o da divisão do espaço de busca em k espaços iguais e selecionando n/k indivíduos de cada espaço e o método da criação a partir de uma população conhecida. A técnica comumente usada é da criação aleatória, por ser mais simples e em linhas gerais, ela gera uma boa distribuição e juntamente com os operadores de mutação, devolvem uma boa exploração do espaço de busca. Diante dos métodos citados, nesse trabalho utilizou-se da criação aleatória para obter resultados iniciais com a aplicação de crossover distintos.

Figura 3: Representação cromossomial adotada nesse trabalho. Fonte: Autor



3.4 Função de avaliação

Conforme afirma [9], a função de avaliação é a maneira utilizada pelos GAs para determinar a qualidade de um indivíduo como solução do problema em questão. Pode-se entendê-la mais facilmente tratando a função de avaliação como uma nota dada ao indivíduo na resolução do problema. Essa nota será usada para a escolha dos indivíduos pelo módulo de seleção de pais.

O foco deste trabalho é em relação à busca da eficiência energética, no entanto, é válido destacar, que para os algoritmos genéticos os melhores indivíduos são os que tem maior probabilidade de sobreviver e transmitir suas características para as próximas gerações, ou seja, os máximos da função. Como o problema em questão é para encontrar as configurações que gaste o mínimo possível de energia atendendo as necessidades dos trens, logo pode-se usar a equação descrita em 2 para encontrar o valor da energia, no entanto para a função será adotada a função 1 que é justamente o inverso da função de energia, sendo assim o algoritmo avaliará os indivíduos de menor gasto de energia como bons indivíduos:

$$g(v) = \frac{1}{f(v)} \tag{1}$$

$$f(v) = \frac{mv^2}{2} \tag{2}$$

onde:
$g(v)$ é a função que o algoritmo irá maximizar
$f(v)$ é a função que mostra o gasto energético
m é a massa do trem
v é a velocidade do trem

3.5 Seleção dos pais

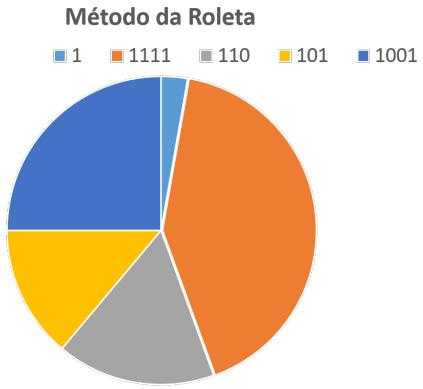
A seleção dos pais deve simular o mecanismo de seleção natural, a qual atua sobre as espécies biológicas, em que os pais mais capazes geram mais filhos, ao mesmo tempo ela permite que os pais menos aptos também gerem descendentes [9]. Nesta etapa, os operadores de seleção devem priorizar os indivíduos que possuem as melhores avaliações, todavia, sem desprezar os indivíduos com baixa avaliação, desse modo, caso sejam selecionados só os melhores, nas próximas gerações, os indivíduos começarão a ficar muito semelhantes e faltará diversidade para que a população possa progredir de forma satisfatória, causando o efeito chamado de **convergência genética**.

3.5.1 Método da roleta

Para melhor entendimento do método da roleta pode-se observar a Figura 4 que possui a tabela com cinco indivíduos e suas respectivas avaliações, no entanto, vale destacar que na prática dificilmente será criada uma população com cinco indivíduos, mas esse exemplo vale para fins didáticos. Então, como foi citado anteriormente, é preciso prover a chance para cada indivíduo. Como descreve [9], nesse método é criado uma roleta (virtual) na qual cada cromossomo recebe um pedaço proporcional à sua avaliação.

Figura 4: Tabela com os indivíduos com suas avaliações e a roleta representando os indivíduos de forma proporcional. Fonte: Autor

Indivíduo	Avaliação
0001	1
1111	15
0110	6
0101	5
1001	9
Total	36



De posse do valor da soma total das avaliações dos indivíduos, pode-se verificar a proporção de cada indivíduo na roleta, como mostrado na Figura 4 No passo seguinte, é sorteado um valor aleatoriamente de 1 até o valor da soma total das avaliações, a fim de se verificar em qual pedaço da roleta ficou o número sorteado e assim descobrir qual o indivíduo que corresponde ao pedaço da roleta.

3.5.2 Método do torneio

O método do torneio como relata [9], consiste em selecionar uma série de indivíduos da população e fazer com que eles entrem em competição direta pelo direito de ser pai. Para selecionar o pai por esse método é necessário determinar o valor do atributo k, que define a quantidade de indivíduos a serem selecionados aleatoriamente dentro da população para competir.

Figura 5: Método do torneio. Fonte: Autor

Método do Torneio para k = 3

Indivíduo	Avaliação
x1	1
x2	15
x3	6
x4	5
x5	9

1	2	3	
x2	x1	x4	x2
x4	x3	x1	x3
x5	x3	x3	x5
x1	x3	x4	x3
x4	x4	x4	x4

A Figura 5 possui duas tabelas onde a primeira possui os indivíduos e suas respectivas avaliações, e a segunda tabela mostra quais os indivíduos k indivíduos escolhidos aleatoriamente. Nesse exemplo foram selecionado 3 indivíduos e entre eles será escolhido o de maior avaliação. Na segunda linha pode-se ver que os indivíduos escolhidos foram o X2, X1 e X4, logo o indivíduo selecionado será o indivíduo X2 por possuir a avaliação 15,

superando a avaliação de X1 e X4.

3.6 *C*rossover

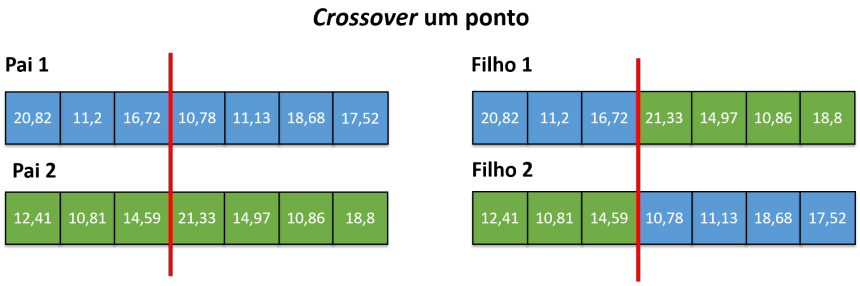
Esse método é baseado no modo como as cadeias de DNA recombinam umas com as outras na reprodução humana para combinar características de cada pai em um filho [10]. Já [11] complementa que nos algoritmos genéticos contínuos há vários modos diferentes de realizar o cruzamento. O cruzamento é dependente da forma como o cromossomo é representado.

3.6.1 *C*rossover de um ponto

O operador de *Crossover* de um ponto como mostrado na Figura 6 é aplicado em dois cromossomos de mesmo comprimento, como a seguir:

- Selecionar um ponto aleatório para o cruzamento
- Separar cada cromossomo em duas partes, dividindo-o no ponto de cruzamento
- Recombinar os cromossomos separados, combinando a parte inicial de um com a parte final do outro e vice-versa, para produzir dois novos cromossomos [12].

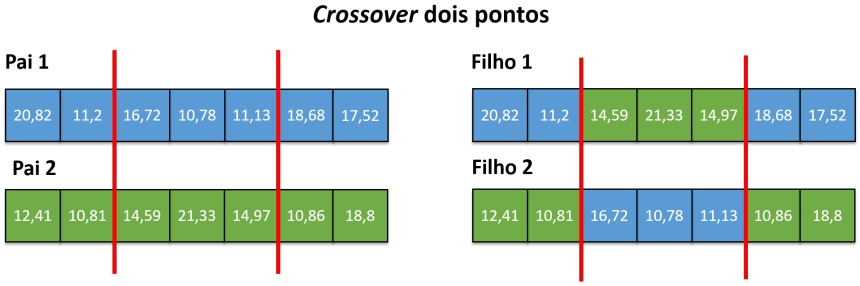
Figura 6: *Crossover* de um ponto. Fonte: Autor



3.6.2 *C*rossover de dois pontos

Segundo [12] nesse *crossover*, dois pontos são escolhidos para dividir os cromossomos em duas seções, com as seções externas juntando-se para transformar o cromossomo em um anel. As duas seções são trocadas como mostrado na Figura 7.

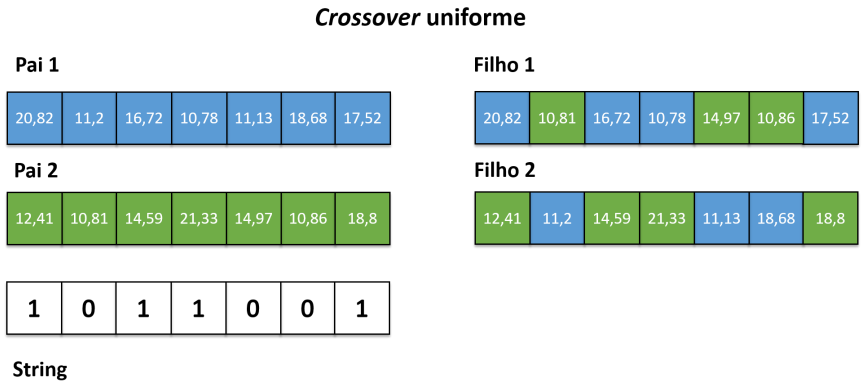
Figura 7: *Crossover* de dois pontos . Fonte: Autor



3.6.3 *C*rossover uniforme ou *c*rossover discreto

Conforme mostrado em [9] o *crossover* uniforme, faz-se um sorteio para escolher em cada posição l um elemento pertencente ao conjunto dado por $\{{c_l}^1, {c_l}^2\}$ e o segundo filho recebe o elemento não sorteado para o primeiro. Um exemplo do funcionamento deste operador pode ser visto na Figura 8. Este operador é mais eficiente quando a função de avaliação é linear, não havendo termos cruzados entre os valores armazenados.

Figura 8: *Crossover* uniforme. Fonte: Autor



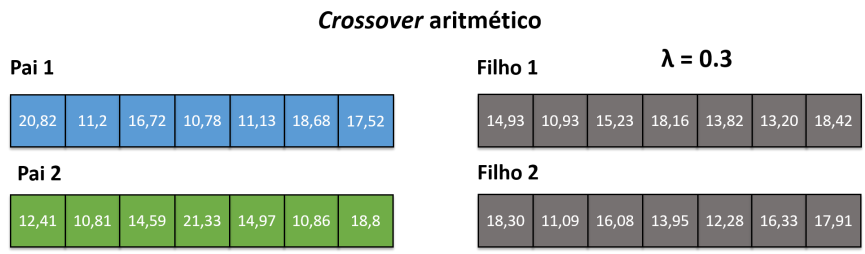
3.6.4 *C*rossover aritmético

Como relata [9] o *crossover* aritmético, define-se um parâmetro $\lambda \in [0, 1]$, e calcula-se cada posição do primeiro filho através da fórmula 3, onde l é o índice da posição que varia de 1 a quantidade de genes dos pais. De maneira análoga, para gerar o segundo filho será usada a fórmula 4.

$$ {c_l}^{filho1} = \lambda {c_l}^1 + (1 - \lambda){c_l}^2 \tag{3} $$

$$ {c_l}^{filho2} = \lambda {c_l}^2 + (1 - \lambda){c_l}^1 \tag{4} $$

Figura 9: *Crossover* aritmético. Fonte: Autor



3.6.5 *C*rossover linear

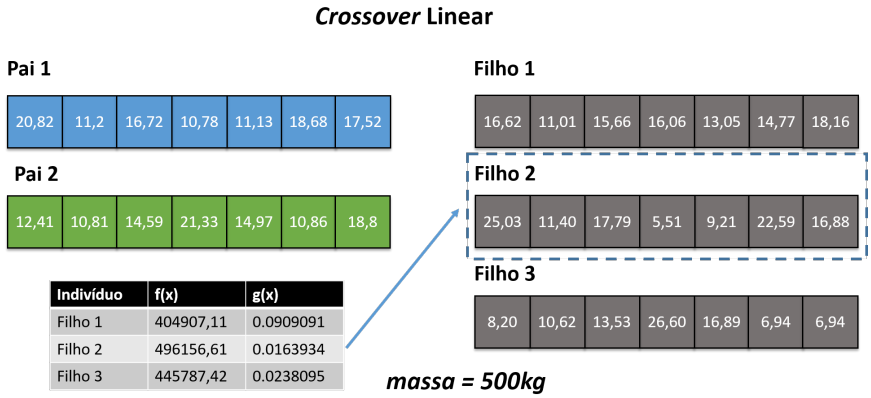
O operador de *crossover* linear conforme apresenta [9] evita a criação de filhos iguais em uma reprodução, de forma que ele gera três filhos com base nas fórmulas 5, 6 e 7. Para manter o tamanho da população é a avaliado cada filho gerado e o filho de avaliação mais baixa é excluído. Como pode ser visto na Figura 10 a tabela que possui o valor da avaliação dos três filhos, sendo que o filho 2 obteve a menor avaliação como mostrado na coluna $g(x)$, logo esse filho será excluído, restando apenas dois filhos.

$$ {c_l}^{filho1} = \frac{{c_l}^1}{2} + \frac{{c_l}^2}{2} \tag{5} $$

$$c_l{}^{filho2} = \frac{3 * c_l{}^1}{2} - \frac{c_l{}^2}{2} \tag{6}$$

$$c_l{}^{filho3} = \frac{-c_l{}^1}{2} + \frac{3 * c_l{}^2}{2} \tag{7}$$

Figura 10: *Crossover* linear. Fonte: Autor

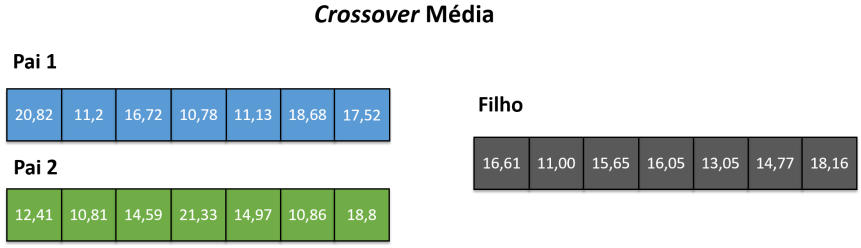


3.6.6 *Crossover* média

Nesse *crossover* é realizado a média entre os valores dos genes de cada pai, ou seja, será feita a média com o primeiro gene do primeiro pai e o primeiro gene do segundo pai, esse processo continua até que seja feita a média entre o último gene do primeiro pai com o último gene do segundo pai, gerando assim um novo valor (caso os genes não sejam os mesmos) para o filho como mostrado na Figura 11. Assim sendo $p_i{}^1$ e $p_i{}^2$ são os cromossomos dos pais, c o cromossomo filho e i um gene do cromossomo. A média aritmética é expressa na equação 8 [11].

$$c_i = \frac{(p_i{}^1 + p_i{}^2)}{2} \tag{8}$$

Figura 11: *Crossover* média. Fonte: Autor

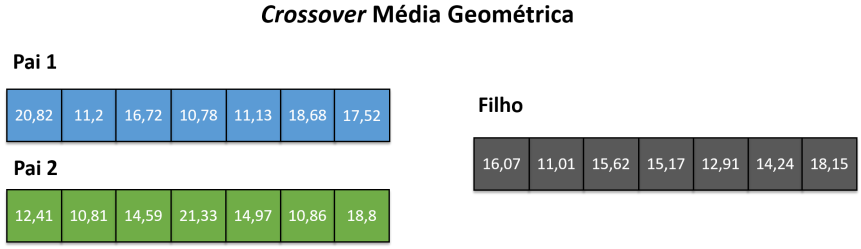


3.6.7 *Crossover* média geométrica

O *crossover* média geométrica, consiste em criar um filho c aplicando a equação descrita em 9, onde $p_i{}^1$ é a informação do gene i do pai n, sendo i a quantidade de genes do cromossomo, e n representa o valor do pai, logo os valores podem ser 1 ou 2. O cromossomo média induz os genes para o meio do intervalo entre os genes, acarretando perda de diversidade [13].

$$c_i = \sqrt[2]{(p_i{}^1 * p_i{}^2)} \tag{9}$$

Figura 12: *Crossover* média geométrica. Fonte: Autor

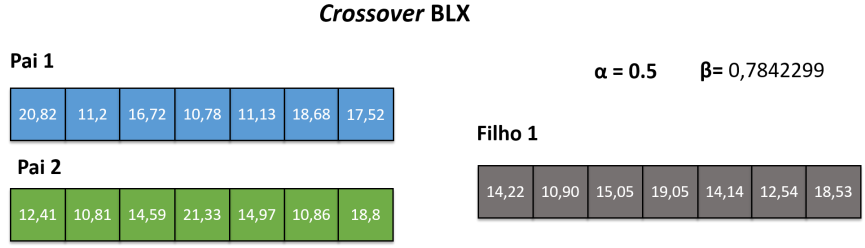


3.6.8 *Simulated Binary Crossover*

Segundo [11] no *Simulated Binary Crossover* (SBX - α) os filhos são criados a partir da fórmula descrita em (10), de forma que o β será um número entre $(-\alpha, 1+\alpha)$. Na Figura 13 é mostrado um exemplo, com $\alpha = 0.5$ e $\beta = 0.7842299$.

$$c = p_1 + \beta(p_2 - p_1) \tag{10}$$

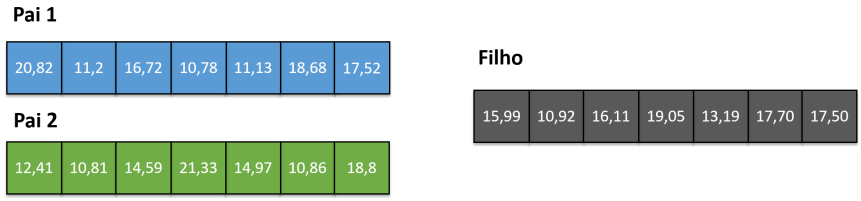
Figura 13: *Simulated Binary Crossover*. Fonte: Autor



3.6.9 *Crossover* flat

O *crossover* flat segundo [9] consiste em estabelecer um intervalo fechado para cada par de valores no cromossomo, do menor valor armazenado até o maior e escolher um valor aleatório pertencente a este intervalo. Pode-se observar o funcionamento desse na Figura 14.

Figura 14: *Crossover* flat. Fonte: Autor

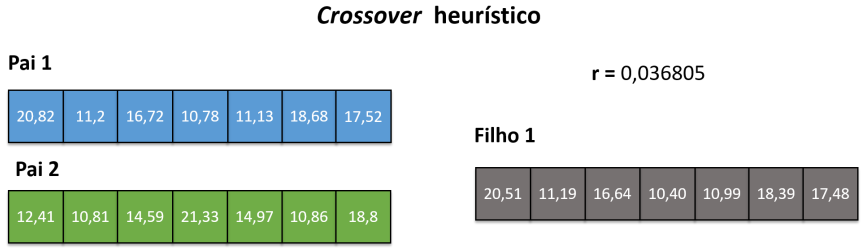


3.6.10 *Crossover* heurístico

O *crossover* heurístico (Figura 15), como descrito por [11] é um *crossover* baseado em direção, onde faz uso da informação da função de avaliação ou do gradiente para determinar a direção da busca. Por exemplo, se temos uma f(x) que é a função de avaliação que desejamos minimizar, o filho é gerado conforme a equação.

$$c = \begin{cases} p_1 + r(p_1 - p_2) & se \quad f(p_1) < f(p_2) \\ p_1 + r(p_2 - p_1) & se \quad f(p_1) \geq f(p_2) \end{cases}$$

Figura 15: *Crossover* heurístico. Fonte: Autor



3.7 Mutação

A etapa final do GA é chamado mutação. A mutação em biologia é relativamente rara, pelo menos na medida em que afeta sensivelmente a prole. Na maioria das implementações de GA a mutação também é rara (na ordem de 2%). A melhor taxa de mutação depende do problema, do tamanho da população, codificação, e outros fatores. Independentemente da sua frequência, a mutação é importante porque permite que o processo evolutivo possa explorar novas soluções potenciais para o problema. Se alguma informação genética está em falta a partir da população, a mutação fornece a possibilidade de injetar essa informação para a população [10].

Logo, pode-se dizer que o operador de mutação é uma heurística exploratória, injetando novos cromossomos na população e permitindo que o GA busque soluções fora dos limites definidos pela população [14]. Um detalhe relevante é que com a representação numérica, não binária, a probabilidade da mutação, deve ser um pouco mais alta, devido ao efeito da ocorrência de duas ou mais mutações no mesmo cromossomo binário [15].

3.7.1 Mutação uniforme

Conforme afirma [11] a mutação uniforme troca o valor de um gene por um número aleatório uniforme no intervalo $[a_i, b_i]$ aceito pelo gene. Caso seja preciso fazer a mutação em um determinado k-ésimo gene p do cromossomo, então o filho c será calculado conforme a equação a seguir:

$$c_i = \begin{cases} U(a_1 + b_i) & i = k() \\ p_i & i \neq k \end{cases}$$

4 Metodologia

Ao fazer uma busca pelos operadores na literatura foi observado que alguns possuíam parâmetros a serem definidos, no entanto, no primeiro momento foram mantidos os valores já usados da fonte, dessa forma podemos citar os seguintes operadores com seus respectivos paramentos.

- No operador de Seleção torneio, é preciso definir qual o valor de K que representa a quantidade de indivíduos que irá ser escolhido para competir, no nosso problema foi usado $K = 3$.
- No operador de *Crossover* aritmético, há a necessidade de definir o valor de λ, onde foi definido como $\lambda = 0.3$.

- O operador *Simulated Binary Crossover* possui o parâmetro α que foi definido como sendo $\alpha = 0.2$.
- O operador *Crossover* heurístico o valor do parâmetro r é um valor aleatório dentro do intervalo (0, 1), mas ele é definido no momento da reprodução.

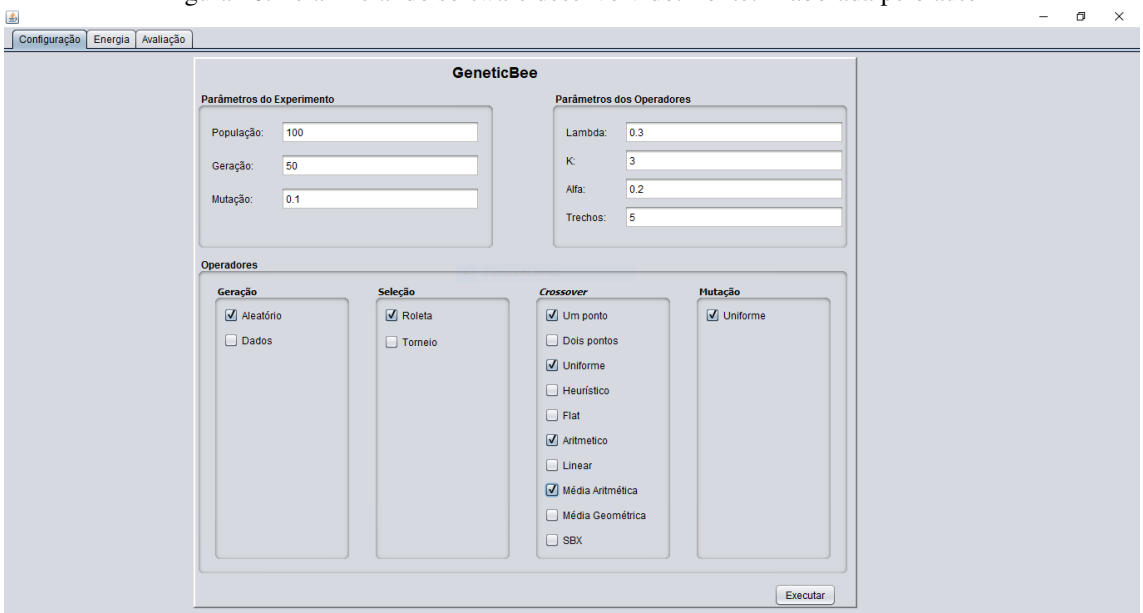
Já os valores em relação a todos os experimentos foram:

- População: 100 indivíduos
- Geração: 50
- Mutação: 0.1

Após realizada a representação cromossômica e as implementações, o próximo passo foi analisar e organizar as implementações, mas de forma que os operadores não interferissem no resultado uns dos outros, visto que havia o interesse em fazer uma comparação, a fim de encontrar os operadores mais adequados para compor o algoritmo genético que busca resolver o problema da eficiência energética dos trens urbanos.

Na Figura 16 pode-se encontrar a tela inicial do software, onde é possível observar elementos gráficos, nos quais o usuário poderá fazer modificações quanto a execução de comparações. Dessa forma, foi criada uma classe Experimento que possui atributos como geração, seleção, *crossover* e mutação. Esses atributos auxiliam no momento de combinar os operadores desejados pelo usuário.

Figura 16: Tela inicial do software desenvolvido. Fonte: Elaborada pelo autor



Já os outros atributos são definidos pelo usuário, como População, Geração e Mutação, que são comuns a todos os experimentos. A cada novo experimento será gerada uma nova população, de forma que cada um seja independente. Após a conclusão dos experimentos, foi gerado gráficos com o auxilio da biblioteca JFreeChart. Essa biblioteca manipula gráficos 2D e 3D e tem fácil integração com Java.

5 Análise dos Resultados

Configurado os parâmetros, foram feitas três execuções do software:

- Na Figura 17 mostra o método da roleta, combinado com todos os tipos de *crossover*

- Na Figura 18, o método do torneio combinado com todos os tipos de *crossover*.
- Na Figura 19 é feita a combinação de todos os operadores implementados.

Figura 17: Resultado com o método da roleta. Fonte: Elaborada pelo autor

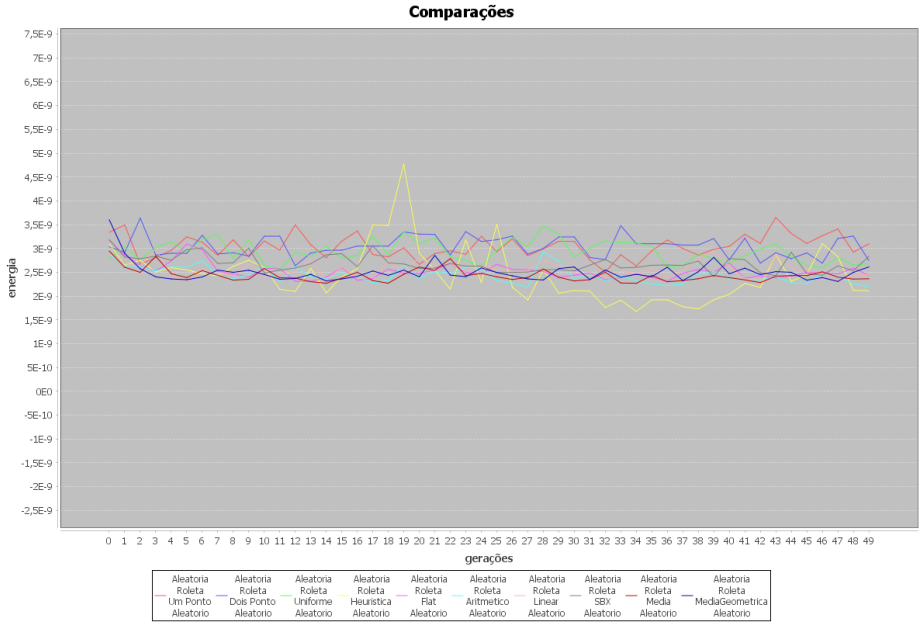
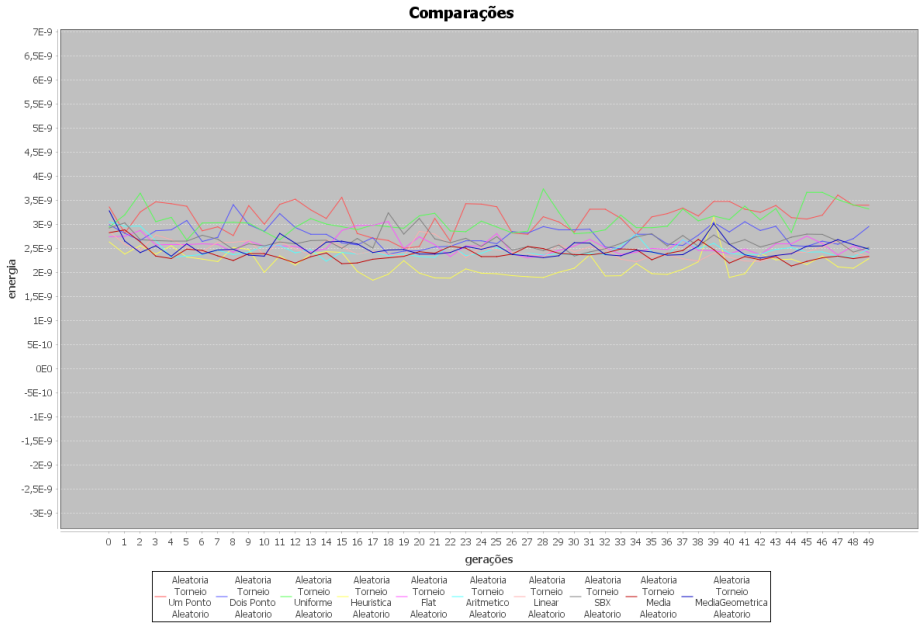


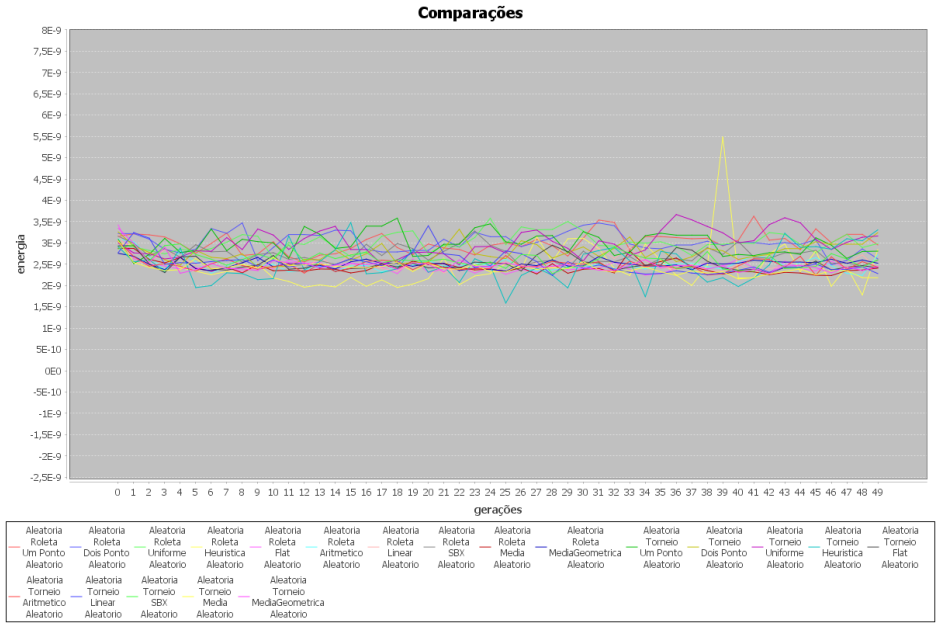
Figura 18: Resultado com o método do torneio. Fonte: Elaborada pelo autor



Nos três gráficos mostrado nas Figuras 17, 18 e 19, é possível observar a relação de numero de gerações pelo valor da energia gasta, ou seja, para cada geração é escolhido o indivíduo que possui o maior valor da avaliação, sendo então colocado no gráfico o valor da energia gasta.

No gráfico da Figura 17 é possível observar que o *crossover* heurístico teve bastante variação, já os *crossover* de média aritmética e média geométrica, se mantiveram mais constantes. No geral, é possível notar que os

Figura 19: Resultado com todos os operadores. Fonte: Elaborada pelo autor



crossover de um ponto, dois pontos e uniforme se mostraram mais eficientes em relação aos outros *crossover*, pois se mantiveram na parte inferior do gráfico, ou seja, a região de menor gasto energético.

O gráfico da Figura 18, mostra os resultados usando o método do torneio. Em geral, os melhores *crossover* foram também os *crossover* de um ponto, dois pontos e uniforme. Apesar de ter sido superior aos demais o uniforme gastou um pouco mais de energia nessa execução. No entanto o *crossover* heurístico obteve o pior desempenho em relação à todos os outros operadores de *crossover*.

Por último, como mostrado na Figura 19, foi combinado todos os operadores, sendo que os dois que mais se destacaram foram os *crossover* de um ponto com o método da roleta e uniforme com o método do torneio, e mais uma vez o método heurístico obteve os piores resultados, pois o mesmo varia muito em relação aos demais.

6 Conclusão

Apesar desse trabalho ser uma parte de um problema maior, que é o da eficiência energética em trens urbanos, o *software* desenvolvido tenta facilitar o processo de aplicação do operador de *crossover*, visto que auxiliará na verificação do comportamento individual de cada *crossover* caso seja inserido uma nova variável, bem como também ajustes na função de avaliação.

Diante do que já foi desenvolvido, é possível destacar os operadores de *crossover* de um ponto e o *crossover* uniforme, como sendo melhores do que os outros operadores comparados nesse trabalho, no entanto conforme o aprimoramento o algoritmo genético atual, pode ser que outro tipo de *crossover* se destaque, visto que os operadores estão relacionados ao problema.

Para os trabalhos futuros pode-se apontar o seguinte melhoramentos:

- Fazer levantamento de trabalhos relacionados na área de análise dos operadores dos GAs.
- Acrescentar as variáveis aceleração e corrente.
- Validar a função de avaliação;
- Inserir dados reais;
- Comparar o resultado do algoritmo genético com os dados reais.

Referências

[1] ONU, disponível em www.onu.org.br/, acessado em 03 de jun de 2016.

[2] LEITE, C. **Cidades inteligentes, Cidade Sustentáveis**. Porto Alegre, Bookman, 2012.

[3] COSTA, Carlos. CIDADES INTELIGENTES E BIG DATA. CADERNOS, **FGV Projetos**,10 , Nº 24, 108-123, 2015.

[4] AQUINO, Andre L.L.; RAMOS, Heitor S.; PEREIRA, Leonardo V.; FRERY, Alejandro C.; **Cidades Inteligentes, um Novo Paradigma da Sociedade do Conhecimento**, p. 165-178 . In: SP: Blucher, 2015.

[5] SOARES, David. **Cidades inteligentes: um novo arranjo para o desenvolvimento.** II Encontro das faculdades de gestão e negócios, Uberlândia, MG, 2012. Disponível em:<http://www.swge.inf.br/PDF/ENFAGEN2012-0100_4886.PDF>. Acesso em: 19 jun. 2016

[6] MARTINS, Marcelle (2015). **Modelo teórico de eficiência energética para sistemas de malhas metroferroviárias**. I Workshop do GPICEEMA. João Pessoa. ISBN: 978-85-237-1027-9, 2015

[7] CBTU, disponível em www.cbtu.gov.br/, acessado em 03 de jun de 2016.

[8] ARAUJO, R. C., Santos, J. L. A., Belo, F. A., Lima, J. A. G; **Utilização do Sistema Telemétrico Dinâmico RailBee para estimativa em tempo real do número de passageiros e análise do desempenho do TUE**. Prêmio Alston de Tecnologia. **Revista Ferroviária** 2010.

[9] LINDEN , Ricardo. **Algoritmos Genéticos**. 3 edição. ed. [S.l.]: Editora Ciência Moderna, 2012.

[10] SIMON, D.**Evolutionary Optimization Algorithms**, 1 edição, editora:Wiley, 2013.

[11] SOUZA, Gustavo. **Otimização de funções reais multidimensionais utilizando algoritmo genético contínuo**, Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, Florianópolis, 2014.

[12] COPPIN, Ben. **Inteligência Artificial**. 1ª Edição. Rio de Janeiro: LTC, 2004.

[13] CARVALHO, E. A. **Sistemas inteligentes: aplicações a recursos hídricos e ciências ambientais.** In: . Porto Alegre: Universidade/UFRGS : Associação Brasileira de Recursos Hídricos., 1999. cap. Capitulo 3:Introdução aos algoritmos genéticos., p. 99,150.

[14] VOSE, M. **The Simple Genetic Algoritm**, Prentice-Hall of India, Nova Dheli, India. 2004

[15] DEB, K. AGRAWAL, S. **Understanding Interaction Among Genetic Algorithm Parameters**, In Banzhaf, W.: Reeves, C> Foundations of Genetic Algoritms 5, pag 256-286, Morgan Kaufman Pub., São Francisco, EUA, 1998

Utilização da robótica com auxílio de técnicas de visão computacional para criação de um dispositivo de detecção de cores primárias

Matheus Aranha Silva[1]
Pablo Henrique da Silva[1]
Eric Gabriel Souza Crespo[1]
Dario Brito Calçada[1]

Resumo: A utilização da robótica tem se mostrado bastante significativo na criação de tecnologias assistivas, conseguindo ampliar algumas habilidades funcionais daqueles que a utilizam. Seguindo esta linha de pesquisa, o presente estudo propõe a criação de um dispositivo para auxiliar portadores de discromatopsia (daltonismo) na identificação de cores primarias utilizando técnicas de robótica e visão computacional. Foram utilizados durante o projeto uma placa Arduino Mega 2560, um sensor de reconhecimento de cor TCS230 e três motores de vibração. O dispositivo se mostrou capaz de captar com eficiência as cores vermelho, verde e azul e converter tais cores ao usuário do dispositivo por meio de sinais vibratórios.

Palavras-chave: Discromatopsia. Robótica. Visão Computacional.

Abstract: *The use of robotics has been quite significant in creating assistive technologies, managing to extend some functional skills of those who use it. Following this line of research, this study proposes the creation of a device to assist people with dyschromatopsia (color blindness) in identifying primary colors using robotic techniques and computer vision. There were used during the project an Arduino board Mega 2560, a TCS230 color recognition sensor and three vibration motors. The device has shown be efficiently to capture the red, green and blue colors and convert these colors to device user through vibratory signals.*

Keywords: Computer Vision. Dyschromatopsia. Robotics.

1 Introdução

A "Discromatopsia" é um termo usado para designar qualquer tipo de defeito de visão de cores. A expressão "daltonismo" é popularmente usada como sinônimo de discromatopsia, em referência ao químico John Dalton, 1766-1844, que tinha protanopia (um tipo de discromatopsia) e foi o primeiro cientista a estudar o assunto [1]. As discromatopsias podem ser congênitas, resultado de alterações genéticas, ou decorrentes de doenças sistêmicas ou oculares [1]. Na população geral, estima-se que as discromatopsias congênitas acometem 6% a 10% dos homens e 0,4% a 0,7% das mulheres [2].

A visão de cores é um fenômeno complexo, que envolve células fotossensíveis especiais, os cones. Na retina humana, existem aproximadamente cinco milhões de cones e cada um contém um tipo específico de fotopsina: vermelha, verde ou azul. A fotopsina é a proteína responsável por converter o sinal luminoso em sinal elétrico, que é conduzido pelo nervo óptico até o córtex cerebral, onde a visão cromática é interpretada. Cada fotopsina é sensível à luz com um comprimento de onda distinto. Em humanos, o mecanismo de visão de cores é fundamentalmente tricromático, pois as diferentes proporções de estimulação dos três tipos específicos de cones explicam todas as outras cores [2][1].

O material genético responsável pela codificação das fotopsinas pertence a uma superfamília de genes que inclui também os genes das proteínas receptoras do odor e do paladar [3]. No locus 7q32.1 encontra-se o gene da fotopsina do cone azul; os genes dos fotopigmentos dos cones vermelho e verde encontram-se no locus Xq28 [4][5]. Os defeitos congênitos da visão cromática são resultado de alterações nos genes codificadores das fotopsinas e são divididos em: tricromatismo anômalo (quando uma das três fotopsinas tem seu espectro de absorção de luz deslocado para outro comprimento de onda), dicromatismo (quando há ausência de um dos tipos de fotopsinas) e monocromatismo (condição muito rara caracterizada pela presença de apenas uma das fotopsinas,

[1] Universidade Estadual do Piauí (UESPI) – Campus Alexandre Alves de Oliveira – Parnaíba – PI - Brazil
{mattwslv,pablohdasilvaluna,erichendrix100,dariobcalcada@gmail.com}

normalmente a azul). Nas situações de tricomatismo anômalo, o defeito de visão cromática costuma ser menos intenso 1231 [5][6]. Ao se nomearem as discromatopsias, por convenção, são usados prefixos gregos para primeiro, segundo e terceiro - “protos”, “deuteros” e “tritos” - para determinar as cores vermelha, verde e azul, respectivamente [1]. Assim, o tricomatismo anômalo pode ser do tipo protanomalia, deuteranomalia ou tritanomalia; e o dicromatismo pode ser do tipo protanopia, deuteranopia ou tritanopia. As situações de defeito do eixo vermelho-verde são as mais frequentes, e estima-se que 5% dos homens sejam deuteranômalos. Defeitos envolvendo os cones azuis são raros, com prevalência de cerca de 1:13.000 [6].

O daltonismo não tem cura, podendo causar problemas para o indivíduo assim como constrangimentos. Várias pesquisas buscam auxiliar o daltônico, como: Semáforos que usam formas geométricas para auxiliar o daltônico a identificar se está vermelho, amarelo ou verde e softwares de correção de cores para a nova geração de iPods e iPhones, que permitem pessoas enxergarem itens que não seriam visíveis naturalmente. Desta forma, exerceriam seus trabalhos de forma mais efetiva e encontrariam melhores soluções para suas limitações [7]. Contudo, a maioria das aplicações que procuram diminuir os efeitos do daltonismo não considera que este distúrbio pode ocorrer em graus variados, diferenciando-se de pessoa pra pessoa [8].

Entre as principais ferramentas usadas nessas pesquisas, podemos destacar a visão computacional e a robótica. A visão computacional é responsável pela visão de uma máquina, pela forma como um computador enxerga o meio à sua volta, extraindo informações significativas a partir de imagens capturadas por câmeras de vídeo, sensores, scanners, entre outros dispositivos. Estas informações permitem reconhecer, manipular e pensar sobre os objetos que compõem uma imagem [9]. Difere do processamento de imagens porque, enquanto ele se trata apenas da transformação de imagens em outras imagens, ela trata explicitamente da obtenção e manipulação dos dados de uma imagem e do uso deles para diferentes propósitos.

A visão computacional tem evoluído rapidamente, produzindo ferramentas que permitem o entendimento das informações visuais, especialmente em cenas com estruturas bem definidas [10].

Dessa forma, a visão computacional fornece ao computador uma infinidade de informações precisas a partir de imagens e vídeos, de forma que o computador consiga executar tarefas inteligentes, simulando e aproximando-se da inteligência humana. Diversos campos estão relacionados à visão computacional. O mais notável é a própria visão biológica, no qual a visão computacional é baseada e que, logicamente, é diretamente relacionado. Na verdade, é justo dizer que essas são áreas análogas, uma para a computação, outra, para a biologia. Afinal, enquanto o último estuda os processos psicológicos envolvidos na formação e percepção de imagens pelos seres vivos, o primeiro estuda os processos e algoritmos usados por máquinas para enxergar. Segundo Jahne e Haubecker [11] a visão computacional é um assunto complexo, pois comparando um sistema técnico, com um sistema biológico, pode-se dividi-la em vários módulos como visualização, formação da imagem, controle da irradiação, entre outros. Assim seu objetivo principal é prover ao computador as capacidades de percepção do sistema visual humano em relação ao ambiente [10].

Enquanto a robótica, uma área multidisciplinar, altamente ativa que busca o desenvolvimento e a integração de técnicas e algoritmos para a criação de robôs, lança mão da tecnologia associada a projeto, fabricação, teoria e aplicação para auxiliar o homem em seus diversos trabalhos [15].

Conforme Groover e seus colaboradores, [12] o primeiro robô móvel construído e reconhecido na bibliografia é o Shakey, desenvolvido pelo Stanford Research Institute, em 1968, e possuía uma variedade enorme de sensores, incluindo uma câmara de vídeo e sensores de toque, binários, e navegava entre as salas do laboratório, enviando sinais de rádio a um computador DEC PDP-10, que permitia efetuar algumas tarefas como, por exemplo, empurrar caixas e evitar obstáculos.

Atualmente devido aos inúmeros recursos que os sistemas de microcomputadores nos oferecem, junto com o grande crescimento tecnológico, os robôs tornaram-se elementos fundamentais na manufatura automatizada. Em função do rápido avanço da automação nas diversas áreas das atividades humanas, a utilização de robôs tem se tornando cada vez mais comum, principalmente em áreas que exigem elevado grau de precisão, confiabilidade e velocidade. Assim o estudo em desenvolvimento nos sistemas de controle dos robôs tem crescido vertiginosamente nos últimos anos, devido a quanto mais complexa a tarefa executada pelo robô, mais complexos são os algoritmos de controle. Por meio do crescimento dos recursos computacionais que surgiram nas últimas décadas foi possível desenvolver robôs capazes de executar tarefas com grau de complexidade superior que os robôs anteriores, que eram restritos apenas a sistemas mecânicos [13].

O presente trabalho teve então como objetivo descrever a criação de uma ferramenta robótica, assistida pela visão computacional, capaz de auxiliar daltônicos na detecção de cores. Realizou-se experimentos em que foi verificado o grau de acerto dos usuários, para encontrar a relação entre usuário com a ferramenta a fim de definir parâmetros de usabilidade. Buscou-se obter dados realizando observações com um grupo de voluntários de forma

a levantar o nível de confiabilidade do aparelho, de acordo com a variação do tempo entre a detecção da cor e a resposta do voluntário.

2 Materiais e Métodos

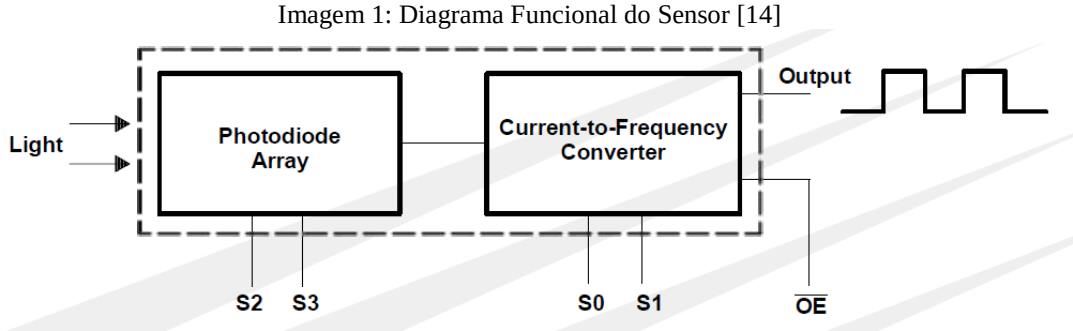
O protótipo da luva de identificação de cores primárias por sensor (L.I.C.P.S.) é composto por uma placa micro controladora Arduino Mega 2560, um sensor de reconhecimento de cor TCS230 e três motores *vibracall* acoplados em uma luva.

2.1 Processo de captação de cor

O processo de captação da cor é realizado pelo sensor de reconhecimento TCS230, composto pela combinação de 64 fotodiodos, sendo 16 fotodiodos com filtro para a cor azul, 16 com filtros para a cor verde, 16 com filtros para a cor vermelha e 16 não possuem filtro.

Os sensores captam a luminosidade do ambiente, filtrando as cores em uma matriz de fotodiodos, a matriz gerada é lida por um conversor de luz para frequência que produz em sua saída uma onda quadrada com frequência diretamente proporcional a intensidade da luz contendo informações sobre a intensidade das cores captadas pelos fotodiodos.

Imagem 1: Diagrama Funcional do Sensor [14]



As informações de intensidade das cores obtidas são enviadas para a placa Arduino Mega 2560 em que o sensor está ligado, que possui um sketch responsável pelo controle e ativação dos motores de acordo com os valores recebidos.

2.2 Representação da cor captada

A representação da cor captada pelo sensor para o voluntário é realizada pela vibração de três motores *vibracall* distintos, acoplados no quinto quirodáctilo (dedo mínimo), quarto quirodáctilo (dedo anelar) e terceiro quirodáctilo (dedo médio) da luva utilizada pelo voluntário.

A ativação dos motores é realizada de acordo com os valores recebidos pelo Arduino, que liga ou desliga um dos três motores conforme o valor que foi recebido pelo sensor.

Tabela 1: Indicação de ativação dos motores de acordo com a cor reconhecida

Cor	Motor Ativado
Vermelho	Motor 1 – Posicionado no quinto quirodáctilo
Verde	Motor 2 – Posicionado no quarto quirodáctilo
Azul	Motor 3 – Posicionado no terceiro quirodáctilo

2.3 Testes para validação

O protótipo foi testado individualmente por dez voluntários aleatórios, sendo realizada uma sequência de três rodadas para adaptação do dispositivo e cinco rodadas para teste do dispositivo com cada um dos voluntários. Os testes tinham como objetivo a verificação da eficiência de reconhecimento das cores primárias do L.I.C.P.S.

No início dos testes foi informado ao voluntário as cores representadas por cada motor e realizado uma rodada de reconhecimento com cada uma das cores (Vermelho, Verde e Azul) para que o voluntário se habituasse com o dispositivo. Após a rodada de reconhecimento, o voluntário era vendado e iniciava-se aos testes aleatórios, onde eram colocados os mesmos objetos da rodada de reconhecimento de forma aleatória e perguntava-se ao voluntário qual a cor do objeto colocado na frente do sensor.

Durante os testes o L.I.C.P.S. permaneceu conectado a um computador utilizando o conector USB tipo B do Arduino ligado a uma porta USB. Sendo possível monitorar as informações sobre a intensidade das cores captadas pelo sensor e comparar com a resposta informada pelo voluntário, analisando se as informações coincidem.

3 Resultados e Discussões

Os testes foram realizados com uma amostra total de 10 voluntários aleatórios, onde cada um dos voluntários foi submetido a um total de 8 rodadas de testes durante o experimento, sendo 3 rodadas iniciais de teste para habituação do voluntário com o dispositivo e 5 rodadas posteriores de testes para avaliar a eficiência do protótipo. Ao total foi obtida uma amostragem de 80 testes, sendo 30 testes para habituação e 50 testes para verificação de eficiência.

Por conta de nenhum dos voluntários não apresentarem nenhum dos tipos de discromatopsia e para que não houvesse nenhum tipo de influência visual em suas respostas durante os testes de verificação de eficiência os voluntários foram submetidos a "testes cegos".

Tabela 2: Resultado dos experimentos realizados

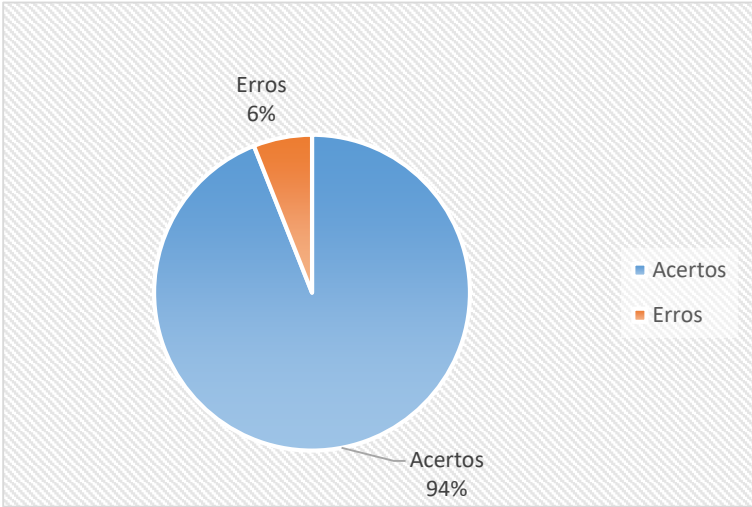
	Teste 1 – (Vermelho)		**Teste 2 – (Verde)**		**Teste 3 – (Azul)**		**Teste 4 – (Verde)**		**Teste 5 – (Azul)**	
Voluntário	Cor Captada	Cor Informada	Cor Captada	Cor Informada	Cor Captada	Cor Informada	Cor Captada	Cor Informada	Cor Captada	Cor Informada
1	Vermelho	Vermelho	Verde	Verde	Azul	Azul	Verde	Verde	Azul	Azul
2	Vermelho	Vermelho	Verde	Verde	Azul	Azul	Verde	Verde	Azul	Azul
3	Vermelho	Vermelho	Verde	Verde	Azul	Azul	Verde	Verde	Azul	Azul
4	Vermelho	Vermelho	Verde	Verde	Azul	Azul	Verde	Verde	Azul	Azul
5	Vermelho	Vermelho	Verde	Verde	Azul	Azul	Verde	Verde	Azul	Azul
6	Vermelho	Verde	Verde	Verde	Azul	Azul	Verde	Verde	Azul	Verde
7	Vermelho	Vermelho	Verde	Verde	Azul	Azul	Verde	Verde	Azul	Azul
8	Vermelho	Vermelho	Verde	Verde	Azul	Azul	Verde	Verde	Azul	Azul
9	Vermelho	Vermelho	Verde	Verde	Azul	Azul	Verde	Verde	Azul	Azul
10	Vermelho	Vermelho	Verde	Verde	Azul	Verde	Verde	Verde	Azul	Azul

Observa-se que, dos 50 experimentos realizados para verificação da eficiência do dispositivo apenas 3 experimentos tiveram respostas diferentes das cores que foram colocadas na frente do dispositivo. Essa diferença entre a cor captada pelo dispositivo e a cor informada pelo voluntário ocorreu por conta do equívoco cometido pelo próprio voluntário, ao assimilar o motor a sua respectiva cor.

A análise dos resultados mostrou um percentual significativo de acertos no reconhecimento das cores submetidas ao protótipo, onde foi obtido um auto rendimento na percepção das cores capturadas pelo sensor, e transpostas para os motores equivalentes da luva, informando corretamente ao usuário qual a cor do objeto analisado, tornando o protótipo um mecanismo de total apoio, capaz de auxiliar na identificação de cores primárias e possibilitando a capacidade de diferenciar ou reconhecer essas cores para pessoas que não possuem tal capacidade.

O percentual do teste pode ser observado no gráfico abaixo, onde obteve-se um total de 94% de acertos e apenas 6% de erros, dada as condições de teste.

Imagem 2: Porcentagem entre acertos e erros



Vale ressaltar que percentual de erro obtido durante os experimentos não foi causada por falha no sistema de reconhecimento das cores ou no sistema de representação das cores captadas para o usuário.

4 Conclusão

O dispositivo conseguiu identificar corretamente em todos os testes as cores dos objetos apresentados e converter os sinais luminosos recebidos em vibração com auxílio dos motores *vibracall*, informando corretamente ao usuário do protótipo qual a cor do objeto inspecionado.

A taxa de acerto significativa de 94% obtida no experimento mostra a possibilidade de utilização da robótica com o auxílio de técnicas de visão computacional, auxiliando a visão humana no reconhecimento de cores primárias, tornando o dispositivo um mecanismo de apoio para portadores dos três grupos discromatopsia: monocromacia, dicromacia e tricomacia anómala.

Um ponto importante a ser tratado futuramente é a utilização de um sensor com melhor captação de cores, ampliando o espectro de reconhecimento. Outro ponto importante a ser tratado futuramente é o aprimoramento da resposta tátil do dispositivo, evitando a confusão de reconhecimento das cores por parte dos usuários.

Referências

[1] BRUNI, L. F.; CRUZ, A. A. V. Sentido cromático: tipos de defeitos e testes de avaliação clínica. Arq. Bras. Oftalmol., São Paulo, v. 69, n. 5, p. 766-775, 2006

[2] GORDON, N. Colour blindness. Public Health, v. 112, n. 2, p. 81-84, 1998.

[3] JACOBS, G.H. Evolution of colour vision in mammals. Philos Trans R Soc Lond B Biol Sci, v. 364, n. 1531, p. 2957-2967, 2009.

[4] NEITZ, M.; NEITZ, J. Molecular genetics of color vision and color vision defects. ArchOphthalmol, v. 118, n. 5, p. 619-700, 2000.

[5] DEEB, S. S. Molecular genetics of colour vision deficiencies. Clin Exp Optom. v. 87, n. 4-5, p. 224-229, 2004.

[6] COLE, B. L. Assessment of inherited colour vision defects in clinical practice. Clin Exp Optom, v. 90, n. 3, p. 157-175, 2007.

[7] CAMPOS E. O daltonismo. Revisa Brasileira de Oftalmologia. 1949; 8:3-27.

[8] GONZALEZ, R., WOODS, R. Processamento de Imagens Digitais. São Paulo: Edgard Blücher, 2000.

[9] BALLARD, Dana Harry, Computer Vision, PrenticeHall, 1982.

[10] SEBE, N.; COHEN, I.; GARG, A.; HUANG, T. S. Machine Learning in Computer Vision. Springer, 2005.

[11] JAHNE, B.; HAUBECKER, H. Computer Vision and Applications. Academic Press, 2000.

[12] GROOVER, Mikell P., WEISS, Mitchell, NAGEL, Roger N. et al. Robótica: tecnologia e programação. São Paulo: McGraw-Hill, 1988. 401p.

[13] PEDROSA, D. P. F., MEDEIROS, A. A D., ALSINA, P. J. (2002). Geração de Caminhos Ponto-a-Ponto para Robôs Móveis com Rodas. Anais do XIV Congresso Brasileiro de Automática.

[14] TAOS, TCS230 Programmable Color Light-to-Frequency Converter, TAOS046, 2003.

[15] SILVA, J. O., MONÇÃO, G. A., CUNHA, N. D., AMARAL, F. R., ROCHA, C. U., FONSECA, A. P., BARROS, K. A. A. L. (2014). Robótica aplicada à saúde: Uma revisão histórica e comparativa da cirurgia robótica. Anais do VIII Fórum FEPEG.