



UNIVERSIDADE FEDERAL DA BAHIA

TRABALHO DE GRADUAÇÃO

**Uma infraestrutura de nuvem de simulação para missões
baseadas em drones**

ANDRÉ VINÍCIUS FARIAS SOUSA

Programa de Graduação em Sistemas de Informação

Salvador
12 de setembro de 2017

ANDRÉ VINÍCIUS FARIAS SOUSA

**UMA INFRAESTRUTURA DE NUVEM DE SIMULAÇÃO PARA
MISSÕES BASEADAS EM DRONES**

Este Trabalho de Graduação foi apresentado ao Bacharelado em Sistemas de Informação da Universidade Federal da Bahia, como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Alírio Santos de Sá

Salvador
12 de setembro de 2017

Ficha catalográfica.

Sousa, André Vinícius Farias

Uma infraestrutura de nuvem de simulação para missões baseadas em drones/ ANDRÉ VINÍCIUS FARIAS SOUSA– Salvador, 12 de setembro de 2017.

96p.: il.

Orientador: Prof. Dr. Alírio Santos de Sá.
Monografia (Graduação)– UNIVERSIDADE FEDERAL DA BAHIA, INSTITUTO DE MATEMÁTICA, 12 de setembro de 2017.

1. computação em nuvem. 2. nuvem robótica. 3. virtualização. 4. veículos aéreos não-tripulados. 5. simulação..

I. Sá, Alírio Santos de. II. UNIVERSIDADE FEDERAL DA BAHIA. INSTITUTO DE MATEMÁTICA. III Título.

CDD XXX.XXXXX

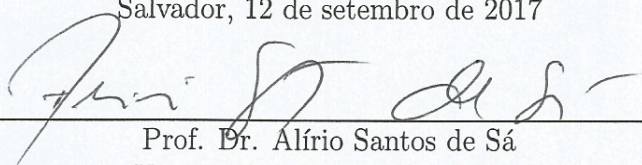
TERMO DE APROVAÇÃO

ANDRÉ VINÍCIUS FARIAS SOUSA

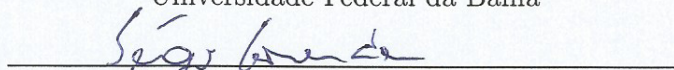
UMA INFRAESTRUTURA DE NUVEM DE SIMULAÇÃO PARA MISSÕES BASEADAS EM DRONES

Este Trabalho de Graduação foi julgado adequado à obtenção do título de Bacharel em Sistemas de Informação e aprovado em sua forma final pelo Bacharelado em Sistemas de Informação da Universidade Federal da Bahia.

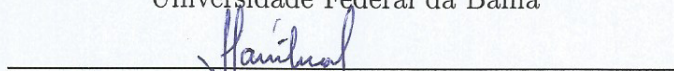
Salvador, 12 de setembro de 2017



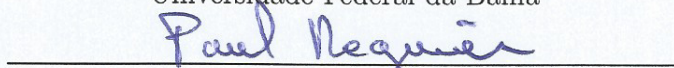
Prof. Dr. Alírio Santos de Sá
Universidade Federal da Bahia



Prof. Dr. Sérgio Gorender
Universidade Federal da Bahia



Prof. Dr. Flávio Morais de Assis Silva
Universidade Federal da Bahia



Prof. Dr. Paul Denis Etienne Regnier
Universidade Federal da Bahia

Dedico esse trabalho a minha filha, agente transformador da minha vida. A minha família, meu alicerce. A minha esposa Lígia, companheira.

AGRADECIMENTOS

Agradeço primeiramente a Deus, por me ensinar a ter foco, atenção e determinação através das graças e também dificuldades que enfrentei no período da graduação.

Sou grato a minha família e amigos por me darem suporte incondicional nos momentos mais importantes da minha vida. Em especial a meus pais. Sim, eu tenho dois, um que me deu a vida e a oportunidade de expressar minha gratidão em por isto enquanto ainda estava em vida. Outro que fez de mim o que sou hoje, que me amou e me ama incondicionalmente e que, me deu um grande irmão ao lado da minha especialíssima mãe, que acaba sempre aparecendo ao meu lado em todos os momentos, bons ou não tão bons assim. A minha pequena grande companheira, Lígia, que esteve ao meu lado durante este período dando apoio em todos os momentos, enfrentando a difícil missão de ter paciência. E como não agradecer a Luisa, minha filha, que de quando era pequenina a mocinha, me incentivou, mesmo sem saber.

Não posso deixar de agradecer também aos professores responsáveis pela minha formação nesta graduação. Aos colegas de curso, que me ajudaram e que, principalmente, deixaram que eu ajudasse, semeando e deixando semear a gratidão. Aos meus colegas de trabalho que sempre tiveram compreensão e me ajudaram muito no decorrer do curso.

Agradeço também ao meu orientador, que me guiou por caminhos complexos e cheios de desafios, mas que me mostraram o mundo da pesquisa e sua importância para a vida.

Sem dúvidas, eu não conseguiria chegar aqui, sem o apoio de todos.

Gratidão para todos.

Só a sinceridade é capaz de resolver os problemas dos indivíduos, do país e do mundo. A deficiência política resulta da falta de sinceridade. A pobreza material e a corrupção moral também têm a mesma origem. Enfim, todos os problemas são gerados pela falta de sinceridade. Religião, Educação e Arte que não se alicerçam na sinceridade passam a representar meras formas sem conteúdo. Homens, a chave de todos os problemas está na sinceridade.

—MOKITI OKADA (1949)

RESUMO

Os *Veículos Aéreos Não-Tripulados (VANTs)* vêm ganhando bastante espaço no dia a dia da sociedade atual, deixando de atender somente ao uso militar para atingir áreas como entretenimento, cinematografia, mapeamento geográfico, missões de busca e salvamento, dentre outras. Mais conhecidos como drones, grandes empresas já realizam testes em atividades cotidianas, como é o caso do serviço de entrega de mercadorias.

Algumas das áreas de utilização dos drones, como o mapeamento geográfico e as missões de busca e resgate, trazem desafios que são enfrentados pela utilização de equipamentos ligados entre si por rede e conectados a uma plataforma computacional para supervisão de missões (Nuvem Robótica). Estes desafios motivam estudos relevantes como detecção de objetos, localização e captura de alvos, planejamento de voo e auto-localização e mapeamento simultâneos de ambientes (Simultaneous Localization and Mapping - SLAM), dentre outros.

O Projeto DaaS (Drone as a Service) tem como objetivo a construção de plataformas, mecanismos e algoritmos que viabilizem a implementação de redes de drones autônomos e colaborativos, podendo ser utilizados sob demanda a partir de uma plataforma de nuvem robótica. O presente trabalho é parte deste projeto e propõe um modelo de Nuvem de Simulação Robótica que fornece ambientes prontos para a simulação de drones. Este modelo entrega ao pesquisador as facilidades que um simulador provê, como redução no custo com aquisição e manutenção de drones, eliminação dos problemas com autonomia da bateria destes equipamentos e possibilidade de testes em ambientes adversos sem riscos, dentre outros, com um tempo de configuração inicial pequeno, acesso ubíquo via rede e recursos computacionais adequados. Assim, esta monografia também apresenta uma implementação desta arquitetura que objetiva atender aos requisitos do projeto. Outra contribuição deste trabalho é um estudo sobre o funcionamento do *Unmanned Aerial Vehicle (UAV)* Parrot AR.Drone 2.0 e seu modelo de tarefas. Este drone é bastante utilizado em pesquisas científicas e está presente no Projeto DaaS.

Palavras-chave: computação em nuvem, nuvem robótica, virtualização, veículos aéreos não-tripulados, simulação

ABSTRACT

The Unmanned Aerial Vehicles (UAV), has a growing presence in modern society at large, expanding its usage out of the military branch, to day to day scenarios, such as entertainment, film making, geographical mapping, search and rescue missions, among others. Better know as drones, big companies already make tests with daily activities such as product delivery.

Some branches where the drones are used, like geographic mapping, and search and rescue operations, have some challenges due to the interconnection among the equipment and the supervisory computing platform for the missions (Robotic Cloud). These challenges motivates relevant studies such as object recognition, location, target capture, flight planning, self locating, and Simultaneous Localization and Mapping (SLAM), among others.

The DaaS project, has as its objective, the construction of platforms, mechanisms, and algorithms that enables the implementation of autonomous, and collaborative, drone network, that can be used by demand trough a robotic cloud platform. This work is a part of this project, and proposes a Robotic Simulation Cloud that provides ready to use environments for drone simulation. This model delivers to the researchers the convenience provided by the simulators, with the cost reduction of acquisition and maintenance of the actual equipment, allowing the creation of custom conditions without risking the drone. This environments are provided in short time and low configuration effort, with ubiquitous network availability and adequate computational capacity. This study also presents an implementation of this architecture that aims to fulfill these requirements. Another contribution of this work is the study of the workings of the UAV Parrot Ar. Drone 2.0 and its task model. This drone is very used in scientific research projects and is present in the DaaS project.

Keywords: cloud computing, cloud robotics, virtualization, unmaned aerial vehicles, simulation

SUMÁRIO

Capítulo 1—Introdução	1
1.1 Motivação	2
1.2 Objetivo	3
1.3 Contribuições	4
1.4 Organização do texto	4
Capítulo 2—Fundamentação Teórica	5
2.1 Computação em Nuvem	5
2.1.1 Definição	5
2.1.2 Características da Computação em Nuvem	6
2.1.3 Modelos de Implementação das Nuvens	6
2.1.4 Modelos de Serviço	7
2.1.5 Benefícios da Computação em Nuvem	9
2.2 Virtualização	10
2.2.1 Arquitetura, Desafios e Soluções	14
2.2.2 Virtualização de Sistemas de Tempo Real	16
2.2.3 RT-Xen	20
2.3 Nuvem Robótica	22
2.3.1 Características	24
2.3.1.1 Independência de Hardware	25
2.3.1.2 Comunicação em Rede Robótica	25
2.3.1.3 Web Services	25
2.3.1.4 Compartilhamento de conhecimentos	25
2.3.1.5 Execução de processamento e armazenamento offboard	26
2.3.2 Desafios	26
2.3.2.1 Escalonamento de Tarefas	26
2.3.2.2 Atraso na comunicação	27
2.3.2.3 Aprendizado de Máquina	27
2.3.2.4 Segurança	28
2.4 Sistema Operacional Robótico	28
2.5 GAZEBO	31
2.6 Plataformas REALabs e REALcloud	33

Capítulo 3—Quadrotor AR.Drone	36
3.1 Introdução	36
3.2 Componentes do AR.Drone	37
3.2.1 O Controlador Central (CC)	38
3.2.2 Sistema de propulsão	39
3.2.3 Verificação de estado da bateria	40
3.2.4 Sistema de Navegação	41
3.2.5 Sistemas Embarcados	42
3.2.6 Comunicação com o cliente	43
3.2.7 Cliente	44
3.3 TUM Simulator - O Simulador do AR.Drone	44
3.3.1 Papel do Robot Operating System (ROS) na simulação do AR.Drone	46
3.3.2 Simulação do AR.Drone no Gazebo	47
3.3.3 Hector Quadrotor Stack	48
3.3.4 Considerações Finais	49
Capítulo 4—Abordagem proposta	50
4.1 Introdução	50
4.2 Trabalhos Correlatos	51
4.3 Nuvem de Simulação e modelo de implementação	52
4.3.1 Interface do Usuário	53
4.3.2 Interface de Programação de Aplicações (API) de Gerenciamento de Ambientes	55
4.3.3 Camada de Validação de Segurança	56
4.3.4 Camada de Gerenciamento	57
4.3.5 Modelos de Ambientes (Templates)	57
4.3.6 Filtro de Segurança	58
4.3.7 Camadas de Virtualização e Simulação	58
4.4 Implementação	60
4.4.1 rtcloud-cli - Gerenciamento de Ambientes	61
4.4.1.1 Elicitação de Requisitos	61
4.4.1.2 Diagrama de Classes	64
4.4.2 Interface do Usuário e API de Gerenciamento de Ambientes	68
4.4.3 Camada de Gerenciamento	69
4.4.4 Camada de Persistência de dados	72
4.4.5 Camada de Validação de Segurança	74
4.4.6 Camada de Virtualização	75
4.4.7 Camada de simulação	76
4.4.8 Filtro de Segurança	78
4.4.9 Rede	79
4.5 Condiserações Finais	80

SUMÁRIO	xi
Capítulo 5—Conclusão e trabalhos futuros	81
Referências	83
Apêndice A—Apêndice	89
A.1 Diagramas de Sequência	89
A.2 Diagrama de Classes	95
Glossário	89

LISTA DE FIGURAS

2.1	Abstração e virtualização aplicadas a discos de armazenamento.	11
2.2	Camadas potenciais para implementação da virtualização.	12
2.3	Modelo de virtualização clássica.	14
2.4	Acesso a dispositivos em domínios paravirtualizados.	21
2.5	Comunicação M2C e M2M.	26
2.6	Relação entre os elementos que compõem o ROS.	30
2.7	Estrutura geral do Gazebo.	32
2.8	Arquitetura da plataforma REALabs.	34
2.9	Arquitetura da plataforma REALabs.	35
3.1	Composição de um quadrotor.	37
3.2	AR.Drone - Camadas de hardware e software	38
3.3	Arquitetura básica do TUM Simulator	45
3.4	Relação de dependência entre os componentes do Simulador AR.Drone. .	46
3.5	Estrutura de controle um AR.Drone real.	47
3.6	Fluxo de dados ROS – GAZEBO.	48
4.1	Arquitetura da Nuvem de Simulação.	53
4.2	Modelo proposto para implementação da Nuvem de Simulação.	54
4.3	Diagrama de sequência: Solicitação do usuário.	56
4.4	Diagrama de implementação da nuvem de simulação do Laboratório de Sistemas Distribuídos (LaSiD).	60
4.5	Diagrama de Casos de Uso do <i>rtcloud-cli</i>	63
4.6	Diagrama de Casos de Uso do escopo definido para o <i>rtcloud-cli</i>	64
4.7	<i>rtcloud-cli</i> - Exemplo do arquivo de parâmetros.	67
4.8	Diagrama de Classes - Carregamento do arquivo de parâmetros.	67
4.9	Diagrama de Classes - Interface do Usuário e API de Gerenciamento. . .	68
4.10	Exemplo de uso do <i>rtcloudcli</i> : Lista de ambientes.	69
4.11	Exemplo de uso do <i>rtcloudcli</i> : Exibe informações de um ambiente.	69
4.12	Diagrama de Classes - Camada de Gerenciamento.	70
4.13	Exemplo de uso do <i>rtcloudcli</i> : Inicialização de um ambiente.	72
4.14	<i>Virtual Machine</i> (VM) executando após inicialização.	72
4.15	Exemplo de uso do <i>rtcloudcli</i> : Parada de um ambiente.	72
4.16	VM não executando após parada do ambiente.	72
4.17	Modelo físico do banco de dados.	73
4.18	Diagrama de Classes - Camada de Persistência de Dados.	74

4.19	Diagrama do modelo <i>Role Based Access Control</i> (RBAC) Básico do National Institute of Standards and Technology (NIST).	75
4.20	Diagrama de Classes do modelo RBAC implementado.	75
4.21	Utilização de processadores virtuais durante a simulação.	78
4.22	Cenário simulado e imagens das câmeras durante a simulação.	79
4.23	Topologia de rede da Nuvem de Simulação.	80
A.1	Caso de uso: Carregar parâmetros do sistema.	90
A.2	Caso de uso: Criar novo ambiente.	91
A.3	Caso de uso: Remover Ambiente.	92
A.4	Caso de uso: Verificar permissão do usuário.	93
A.5	Caso de uso: Executar perfil de firewall.	94
A.6	Caso de uso: Visualizar perfil de firewall	95
A.7	Diagrama de Classes do <i>rtcloud-cli</i>	96

LISTA DE TABELAS

3.1	Tarefas do sistema de propulsão.	40
3.2	Tarefas do sistema de monitoramento da bateria.	41
3.3	Tarefa principal do sistema de navegação.	42
3.4	Pacotes componentes da pilha Hector Quadrotor.	48
4.1	Requisitos funcionais do sistema <i>rtcloud-cli</i>	65
4.2	Requisitos não funcionais do sistema <i>rtcloud-cli</i>	66
4.3	Padrões de projeto adotados no <i>rtcloud-cli</i>	66
4.4	Descrição de comandos aceitos pelo sistema e suas respectivas classes de comando.	69

INTRODUÇÃO

Os *Veículos Aéreos Não-Tripulados (VANTs)*, também conhecidos como *Unmanned Aerial Vehicle (UAV)* ou, mais popularmente, como *drones*, tiveram seu desenvolvimento iniciado objetivando o militar no início do século XX e a partir da década de sessenta este desenvolvimento foi bastante acelerado (Cook, 2007). Atualmente, os drones têm alcançado uma utilização muito variada na sociedade, como o mapeamento geográfico, a cinematografia, o entretenimento, missões de busca e salvamento etc. Os desafios de algumas dessas utilizações, como mapeamento geográfico e missões de busca e salvamento, são atacados pela utilização de UAVs ligados em rede entre si e conectados a uma plataforma computacional para supervisão de missões (Nuvem Robótica)¹, atraindo assim a atenção de pesquisadores e produzindo muito conhecimento.

Conectar UAVs à Nuvem Robótica permite ampliar os limites computacionais e de armazenamento dos UAVs, pois cargas computacionais complexas podem ser executadas em servidores com maior poder de processamento, tendo impacto direto tanto no custo de produção como na autonomia de voo. Mais do que isto, missões mais complexas podem ser realizadas por times de drones, o que tem sido motivador para a pesquisa de algoritmos para área de detecção, localização e captura de alvos, planejamento de voo, auto-localização e mapeamento simultâneos de ambientes, conhecido como Simultaneous Localization and Mapping (SLAM), dentre outros (Hu et al., 2012).

A utilização de simuladores durante estas pesquisas permite que os algoritmos sejam testados e amadurecidos para só então executarem em drones reais, diminuindo o risco de acidentes. Os limites impostos pelo espaço físico, como disponibilidade de rede em ambientes abertos, pouco espaço disponibilizado em laboratórios e restrições de horário também são superados. Por fim, a possibilidade de se realizar testes com diversos modelos de drones, sensores e atuadores sem a necessidade de adquiri-los ou instalá-los diminuem os custos e riscos do projeto.

¹Os robôs ligados em rede, ou Networked Robotics, e as Nuvens para aplicações robóticas, ou Cloud Robotics, são discutidos na Seção 2.3.

Este trabalho apresenta uma Nuvem de Simulação Robótica que fornece ambientes virtualizados independentes para a simulação de quadrotoros como serviço para pesquisadores integrantes de projetos de pesquisa. É descrita uma arquitetura distribuída baseada em máquinas virtuais capazes de executar tarefas de tempo real e que disponibiliza um ambiente completo para a execução de simulações robóticas. Também é apresentada uma implementação desta arquitetura que busca atender aos requisitos das pesquisas sobre navegação autônoma de UAVs realizadas no Laboratório de Sistemas Distribuídos (LaSiD)². Nesta implementação foi utilizado o gerenciador de máquinas virtuais Xen (Barham et al., 2003) com o escalonador de tempo real RT-Xen (Xi et al., 2014). As máquinas virtuais fornecidas pelo ambiente disponibilizam o framework Robot Operating System (ROS) (Quigley et al., 2009) como plataforma de desenvolvimento para aplicações robóticas, o Gazebo (Koenig e Howard, 2004) para a simulação de robôs e ambientes dinâmicos e o *TUMSIMULATOR* (Huang e Sturm, 2014) para a simulação do quadrotor AR.Drone (Krajník et al., 2011; Bristeau et al., 2011) que é o equipamento atualmente utilizado pelo LaSiD em seus experimentos.

1.1 MOTIVAÇÃO

Este trabalho pertence ao escopo do Projeto DaaS, cujo objetivo é o enfrentamento do desafio de desenvolver serviços confiáveis e inteligentes de provisão de recursos computacionais em plataformas cibernéticas físicas de computação em nuvem. Estes serviços são utilizados no suporte às aplicações cibernéticas-físicas de sensoriamento e aquisição de dados baseadas em redes de VANTs. As pesquisas realizadas no projeto envolvem a utilização de microdrones com baixo poder computacional e autonomia energética, a exemplo dos UAVs modelo AR.Drone utilizados no projeto. A realização de experimentos com estes microdrones traz as seguintes dificuldades:

- a) Os pesquisadores envolvidos nos projetos do laboratório utilizam drones reais para seus experimentos, sendo que cada equipamento é utilizado exclusivamente por um pesquisador, o que acarreta limites de pesquisadores ou possíveis conflitos de horário caso haja compartilhamento de equipamento.
- b) A duração dos experimentos realizados está limitada ao uso das baterias que duram poucos minutos. Para melhorar este cenário foram adquiridas baterias adicionais, mas não são suficientes para um período mais prolongado de experimentos (várias horas, por exemplo);
- c) Os experimentos são realizados nas dependências do laboratório, logo a área física é limitada sendo necessário utilizar o ambiente externo para a realização de testes que necessitem mais espaço. Isto insere mais algumas dificuldades como falta de rede cabeada para acesso à rede da instituição caso necessário, maior poluição do sinal de rede sem fio e maior risco de acidentes com o equipamento, que está exposto a vento, animais voadores e outros aspectos naturais;

²A página oficial do LaSiD está disponível em <http://www.lasid.ufba.br>.

- d) O pesquisador precisa deslocar-se até o laboratório sempre que for utilizar o drone, restringindo o horário/tempo de pesquisa, bem como o apoio de outros pesquisadores que estejam em outras localidades.

A utilização de simuladores atende a estas dificuldades, contudo, fazer sua instalação e configuração pode exigir um conhecimento técnico que o pesquisador não possui naquele momento, gastando assim um tempo de aprendizado que poderia ser revertido para a pesquisa. Os requisitos computacionais do desktop utilizado pelo pesquisador podem também não ser suficientes para simular todos os cenários, aumentando também os custos para aquisição de equipamentos do projeto. Uma arquitetura em nuvem supera estes desafios fornecendo em poucos minutos ambientes prontos para uso com poder computacional suficiente para atender às demandas de simulação.

Este trabalho tem como motivação o enfrentamento do desafio de desenvolver um ambiente de simulação de drones em nuvem, contribuindo assim para a diminuição dos riscos, custos de aquisição e manutenção dos equipamentos e, também, para o aumento da mobilidade e produtividade dos pesquisadores, diminuindo o tempo total da pesquisa. Nuvens de simulação já estão disponíveis para uso na Internet, contudo, possuem um custo que pode não ser facilmente absorvido por um projeto. Além disto, quesitos referentes a flexibilidade, como o uso de simuladores específicos ou a não possibilidade de simular componentes relacionados com os robôs, como a rede de comunicação também motivam o desenvolvimento de uma nuvem de simulação para atender às demandas dos pesquisadores.

1.2 OBJETIVO

O presente trabalho tem como objetivo construir uma arquitetura em nuvem que forneça ambientes de simulação adequados às pesquisas de voos autônomos com quadrotores realizadas no LaSiD³. O projeto deve disponibilizar ambientes que permitam aos pesquisadores planejar simulações com alto nível de fidelidade em relação aos ambientes e equipamentos reais.

Os seguintes objetivos específicos foram definidos para alcançar o objetivo principal deste trabalho:

- a) Realizar uma pesquisa exploratória sobre Nuvens Robóticas e tecnologias associadas, identificando os trabalhos existentes nesta área e seus desafios;
- b) Projetar um modelo conceitual inicial da Nuvem de Simulação, baseado na pesquisa realizada, que atenda às necessidades do LaSiD;
- c) Mapear o funcionamento e modelo de tarefas do quadrotor utilizado no LaSiD;
- d) Projetar um modelo de implementação utilizando Software Livre aderente aos equipamentos disponíveis para o LaSiD;

³Para esta monografia quando mencionadas as pesquisas do LaSiD serão consideradas apenas as pesquisas que envolvem quadrotores.

- e) Realizar uma pesquisa exploratória sobre simuladores que possam atender ao projeto DaaS.
- f) Implementar e testar o modelo de implementação desenhado.

1.3 CONTRIBUIÇÕES

No trabalho apresentado por esta monografia destacam-se as seguintes contribuições:

- Foi realizado um estudo sobre as Nuvens Robóticas e as principais tecnologias envolvidas em sua construção, apresentando uma visão geral sobre suas características, desafios e soluções;
- Foi realizado um estudo onde o UAV AR.Drone 2.0 foi explorado, sendo apresentado o seu modelo de tarefas juntamente com algumas características de funcionamento;
- Foi proposto um modelo de arquitetura para uma Nuvem de Simulação Robótica, incluindo um modelo para implementação dessa arquitetura e uma implementação deste modelo utilizando software de código fonte aberto;
- Foram produzidos artefatos de engenharia de software que servem como base para novas implementações do sistema de gerenciamento da nuvem no modelo proposto.

1.4 ORGANIZAÇÃO DO TEXTO

Os demais capítulos desta monografia estão distribuído conforme descrito a seguir. O **Capítulo 2** apresenta os conhecimentos fundamentais relacionados com o presente trabalho e que são parte fundamental da Nuvem de Simulação Robótica aqui proposta. No **Capítulo 3**, o UAV AR.Drone e seu simulador são apresentados. No **Capítulo 4** a arquitetura desta Nuvem é descrita e, também, é apresentada uma implementação baseada em Software Livre desta arquitetura. O **Capítulo 5** apresenta as considerações finais acerca deste trabalho e as sugestões para trabalhos futuros. Por fim, é apresentado o apêndice contendo diagramas de classes e sequência citados no texto.

FUNDAMENTAÇÃO TEÓRICA

2.1 COMPUTAÇÃO EM NUVEM

2.1.1 Definição

Apesar da ideia de recursos computacionais estarem disponíveis como um serviço público existir desde a década de 60, o termo Computação em Nuvem (Cloud Computing) só foi mencionado no final dos anos 90, pelo prof. de sistemas de informação Ramnath Chellapa, em sua palestra “Intermediaries in Cloud-Computing: A New Computing Paradigm” e começou a tornar-se uma realidade recentemente com o surgimento da Salesforce.com, o primeiro modelo de software como serviço disponibilizado via Internet em 1999 (Castro et al., 2013). Encontrar uma definição única para a Computação em Nuvem ainda é um trabalho que está em andamento, existindo assim na literatura muitas definições com diferente pontos de vista (Zhang et al., 2010). Vaquero et al. (2008) apresenta um trabalho que reflete bem esta realidade, comparando mais de 20 definições diferentes. O presente trabalho adota a seguinte definição, apresentada pelo National Institute of Standards and Technology (NIST) em 2009 e revisado em 2011, que é bem aceita pela comunidade científica e reúne os principais aspectos relevantes sobre o tema (Mell e Grance, 2011):

Computação em Nuvem é um modelo que permite o acesso ubíquo, conveniente e sob *demanda* via rede a um conjunto de recursos computacionais configuráveis (ex.: redes, servidores, sistemas de armazenamento, aplicações e serviços) que podem ser provisionados rapidamente e disponibilizados com o mínimo de esforço gerencial ou dependência do provedor de serviços¹.

Nas próximas seções serão discutidas as características, modelos de serviços e modelos de implementação definidos pelo NIST e outros autores.

¹Tradução livre.

2.1.2 Características da Computação em Nuvem

Muitas características são descritas pelos autores envolvidos com o desenvolvimento da Computação em Nuvem. Mell e Grance (2011) e Armbrust et al. (2009) sintetizam em seus trabalhos algumas características que são aceitas por muitos trabalhos acadêmicos e presentes nos provedores de serviços em nuvem:

- Provisionamento de recursos computacionais realizado pelo próprio usuário (*self-service*) de acordo com as suas necessidades (on demand), sem dependência do provedor de serviço.
- Envolve diversas tecnologias que formam um conjunto de recursos que são virtualizados, compartilhados entregues ao usuário dinamicamente. Aspectos como a localização física dos recursos utilizados (país, continente, etc) não são controlados pelo usuário, sendo assim independente. Em alguns modelos de negócio é permitido que o usuário faça esta escolha.
- Os recursos estão disponíveis através da rede, podendo ser acessados por diversos tipos de dispositivos (telefones inteligentes, computadores portáteis, estações de trabalho, dentre outros).
- Tem facilidade na alocação e liberação dos recursos, permitindo seu uso sob demanda (elasticidade) e passando a impressão de que seus usuários têm uma quantidade ilimitada de recursos à sua disposição no momento em que precisarem. Esta alocação e liberação é realizada automaticamente em algumas abordagens de nuvem;
- Traz um modelo de bilhetagem por fração de recurso, permitindo o pagamento por utilização em prazos curtos (ex.: por hora ou por dia). Este modelo permite que seus usuários realizem um investimento baixo para obterem os recursos necessários à sua demanda inicial e aumentem estes recursos apenas quando necessário.

2.1.3 Modelos de Implementação das Nuvens

Dentre os tipos de implementação de uma nuvem que são descritas na literatura, quatro estão mais presentes: Nuvens Públicas, Nuvens Privadas, Nuvens Híbridas e Nuvens Privadas Virtuais, os quais são descritos a seguir:

a) Nuvens Públicas (Public Clouds)

São nuvens que estão abertas para utilização do público em geral. Podem ser operadas, gerenciadas e pertencerem a empresas, organizações governamentais ou universidades (Mell e Grance, 2011). As nuvens públicas são disponibilizadas sob o modelo pague pelo que utiliza (Armbrust et al., 2009), evitando um investimento inicial alto em infraestrutura e transferindo o risco da operação para o provedor da nuvem (Zhang et al., 2010).

b) Nuvens Privadas (Private Clouds)

São nuvens onde sua infraestrutura está disponível exclusivamente para uma única organização. Pode pertencer e ser gerenciada e operada pela própria organização, por uma organização terceira ou ambos (Mell e Grance, 2011). As nuvens privadas oferecem um alto grau de controle. Em contrapartida, demanda um investimento inicial alto e mantêm o risco da operação com a organização (Zhang et al., 2010).

c) Nuvens Comunitárias (Community Clouds)

São nuvens cuja utilização é exclusividade de uma comunidade de usuários que compartilham interesses específicos, que podem ser políticas, requisitos funcionais, objetivos, etc. Estas Nuvens podem ser construídas e mantidas tanto por membros da comunidade quanto por terceiros (Mell e Grance, 2011), além de ser uma alternativa às Nuvens Públicas para casos nos quais estas não possam ser aplicadas. Um exemplo seria a criação de uma Nuvem governamental para atender a leis e normas de segurança que protegem informações governamentais.

d) Nuvens Híbridas (Hybrid Clouds)

São nuvens compostas de múltiplos tipos de nuvem (por exemplo, privada e pública) que permitem a portabilidade de dados entre si (Mell e Grance, 2011). Com as nuvens híbridas, as organizações podem focar seu parque computacional no processamento das tarefas que são mais vitais para seu negócio e necessitam mais segurança e acompanhamento, enquanto que as tarefas secundárias podem ser processadas em uma nuvem pública. As nuvens híbridas fizeram com que a pesquisa e a adoção de padrões para a computação em nuvem ganhasse força (Dillon et al., 2010).

e) Nuvens Privadas Virtuais (Virtual Private Clouds)

São nuvens públicas que criam em seu topo uma camada de rede virtualizada utilizando a tecnologia *Virtual Private Networks* (VPN), o que permite a um provedor de serviços desenhar sua própria topologia e esquema de segurança de rede de forma a também utilizar a sua infraestrutura privada interligada com a infraestrutura da nuvem pública (Zhang et al., 2010). Por manter o modelo de negócio *pay-as-you-go*, as Nuvens Privadas Virtuais oferecem um balanceamento perfeito entre o controle das nuvens privadas e a flexibilidade das nuvens públicas (Dillon et al., 2010).

2.1.4 Modelos de Serviço

A Computação em Nuvem não se restringe às mudanças tecnológicas, mas também à forma como os produtos oriundos da computação são ofertados aos consumidores. O modelo de entrega de produtos ou valores como serviço é a base desta mudança trazendo duas decisões importantes às instituições:

- a) **Adquirir ou contratar?** Como consumidores, as organizações precisam agora inserir em suas decisões de custos a possibilidade de contratar muitos serviços de

TI que antes não estavam disponíveis sob este modelo de negócio ao invés de adquiri-los e mantê-los em seus datacenters.

- b) **Vender ou fornecer como serviço?** Como fornecedores, as organizações também precisam inserir no seu modelo de negócios a possibilidade de fornecer seus produtos como serviço, e isto pode acarretar mudanças profundas na estrutura da organização.

Os três modelos de serviço principais são: Infraestrutura como Serviços (IaaS, Infrastructure as a Service), Plataforma como Serviços (PaaS, Platform as a Service), Software como Serviço (SaaS, Software as a Service) (Mell e Grance, 2011; Taurion, 2009; Dillon et al., 2010), os quais são descritos a seguir:

- a) **Infrastructure as a Service (IaaS):** O modelo IaaS fornece aos consumidores ambientes computacionais compostos por processamento, armazenamento e rede (dentre outros recursos) para que os sistemas necessários sejam instalados. Esta é a camada mais básica ofertada pelas nuvens, onde o cliente tem o controle a partir da camada de sistema operacional e, eventualmente, algumas opções de rede. Geralmente o produto entregue pelo provedor de serviços em um modelo IaaS é um servidor virtual, em alguns casos com o Sistema Operacional instalado. O principal público alvo deste modelo são as organizações que já possuem uma infraestrutura de TI montada e desejam diminuir seus custos operacionais migrando-os para a nuvem ou novos negócios que necessitam de um investimento inicial em infraestrutura de TI. Alguns exemplos de provedores de serviços IaaS são o Amazon EC2, Rackspace CloudServers e Microsoft Azure.
- b) **Platform as a Service (PaaS):** O modelo PaaS fornece uma plataforma para que os clientes instalem aplicações desenvolvidas ou adquiridas por eles que sejam suportadas pelo provedor de serviços. Alguns provedores fornecem também um ambiente capaz de absorver todo ciclo de vida da aplicação desde o desenvolvimento até seu descarte. O cliente possui controle apenas das aplicações instaladas por ele, sem qualquer gerência sobre a infraestrutura que está abaixo desta plataforma (servidores, armazenamento, rede, etc). O principal público alvo deste modelo de serviço são desenvolvedores de software, organizações que desenvolvem suas próprias aplicações ou que possuem interesse na aquisição de software desenvolvido para a nuvem. São exemplos deste modelo de serviço o Google AppEngine e Windows Azure e Force.com.
- c) **Software as a Service (SaaS):** O modelo SaaS fornece aos clientes uma determinada aplicação ou instâncias desta, que pode ser configurada para se adequar à sua necessidade. Os clientes não possuem qualquer gerência ou controle da plataforma sob a qual a aplicação executa, como servidores, armazenamento, rede ou Interfaces de Programação de Aplicações (APIs) de desenvolvimento. Todo o gerenciamento do ciclo de vida da aplicação e seus custos é de responsabilidade do provedor de serviços, diminuindo os custos do cliente com equipamentos, licenças de software

e profissionais especializados e permitindo com que a organização direcione estes esforços para o objetivo-fim do negócio. São exemplos deste modelo de serviço o Google Docs, Salesforce.com e Microsoft Office 365.

Apesar destes três modelos principais estarem associados à definição de computação em nuvem, existe uma tendência de se ofertar tudo que possa ser entregue dentro do paradigma de Computação em Nuvem como um serviço, o que deu origem aos termos "Everything as a Service (EaaS)" e "X as a Service (XaaS)" frequentemente vistos na literatura mais atual acerca da Computação em Nuvem. [Duan et al. \(2015\)](#) pesquisou as diversas propostas acadêmicas pra novos modelos desde 1984, classificando e tipificando estas propostas e demonstrou os muitos avanços obtidos pela área de computação como serviço e que a tendência atual é de crescimento das pesquisas nesta área.

2.1.5 Benefícios da Computação em Nuvem

Não é por acaso que muitas organizações estão incorporando a Computação em Nuvem à sua operação de TI. Os benefícios que atraem as empresas para este novo paradigma da TI ultrapassam o campo técnico e atingem dois pontos fundamentais para qualquer organização: diminuição nos custos e ampliação do modelo de negócio. [Armbrust et al. \(2009\)](#), [Zhang et al. \(2010\)](#) discutem os benefícios mais relevantes:

- a) **Amplo acesso aos serviços via rede.** A possibilidade de acesso global aos serviços a qualquer hora através de diversos tipos dispositivos (telefones, tablets, computadores portáteis, estações de trabalho e até dispositivos de jogos eletrônicos) permite que a organização tenha um alcance mais amplo principalmente na realização de novos negócios. A criação de escritórios remotos em outros países também é bastante facilitada pelo acesso global que as Nuvens oferecem.
- b) **Redução de custos com a economia de escala.** Os benefícios da economia de escala permitem que o custo direto e indireto das tarefas processadas nos sistemas computacionais tenham quedas expressivas. Os muitos clientes que compartilham a infraestrutura dos provedores de serviços na Nuvem geram uma alta demanda de equipamentos, energia, arrefecimento e sistemas de telecomunicações, dentre outros, e possibilitam uma economia na ordem de 1/3 a 1/7 do custo que um datacenter de médio porte teria. Isto possibilita aos provedores de serviço oferecerem unidades computacionais mais baratas, cobradas por tempo de uso.
- c) **Redução em custos de aquisição e manutenção.** A contratação de serviços em Nuvem transforma os custos de capital em custos operacionais mais baratos, sem a necessidade de altos gastos com manutenção do espaço físico, capacitação de profissionais especializados e contratos de manutenção.
- d) **Pague pelo que utiliza.** O modelo de bilhetagem das Nuvens permite que as organizações paguem apenas pelo que efetivamente utilizarem, facilitando a entrada de novas empresas ou produtos no mercado, uma vez que o investimento inicial em equipamentos pode ser direcionado para outras áreas de negócio, como o marketing ou vendas e aumentado assim que novos clientes sejam captados.

- e) **Processamento paralelo de cargas de trabalho pesadas.** O custo com a contratação de muitos servidores por poucas horas pode ser mais interessante do que utilizar um grande servidor por muitas horas. Sendo assim, uma aplicação que se beneficia do processamento paralelo (ex.: processamento em lote ou análise de dados) pode realizar seu trabalho em um menor tempo, dando mais agilidade para o negócio.
- f) **Agilidade na ampliação de recursos.** A elasticidade e escalabilidade características da Computação em Nuvem permite com que tarefas ou serviços que possuem grande oscilação em sua demanda mantenham um bom desempenho, uma vez que os recursos necessários estarão disponíveis de forma rápida. Alguns provedores disponibilizam, inclusive, a opção de ampliação e redução de recursos de forma automática de acordo com a demanda existente.
- g) **Disponibilidade do serviço.** Os provedores de serviços em Nuvem possuem como objetivo um alto índice de disponibilidade. Um exemplo bem prático é dado por [Armbrust et al. \(2009\)](#): “Se as pessoas tentam fazer uma busca no Google e este está indisponível, elas acreditam que sua conexão com a Internet está com problemas”. De fato, os provedores de serviço investem valores substanciais em medidas que aumentam a disponibilidade dos seus serviços, inclusive com a construção de datacenters em continentes diferentes.

Apesar dos muitos benefícios, a migração de serviços para a nuvem pode envolver um esforço considerável e isto também deve ser levado em consideração junto com outros fatores, como confidencialidade dos dados, custo com links para a transferência dos dados para a Nuvem, dentre outros.

2.2 VIRTUALIZAÇÃO

Dentro da computação, a virtualização é um conceito muito abrangente que há bastante tempo é utilizado nos sistemas computacionais e envolve diversas tecnologias. A virtualização cria uma camada entre o recurso real de hardware ou software e os usuários destes recursos, diferenciando a arquitetura implementada nas camadas inferiores do comportamento percebido nas camadas superiores à de virtualização ([Chiueh e Brook, 2005](#); [Figueiredo et al., 2005](#); [Smith e Nair, 2005](#)). Esta técnica já é utilizada nos sistemas computacionais para se obter uma melhor utilização de recursos. O mecanismo de memória virtual apresenta aos processos uma quantidade de memória maior do que a existente fisicamente, enquanto seleciona quais páginas vão permanecer em memória ou serão gravados em disco. Os processadores mais modernos otimizam sua utilização implementando núcleos lógicos que executam conforme a técnica de compartilhamento de tempo, apresentando aos sistemas operacionais mais núcleos do que os existentes fisicamente. A partir dos anos 2000 tornou-se mais popular a virtualização de todos os componentes de um computador e suas diversas camadas de abstração, resultando em objetos virtuais que variam de computadores completamente virtuais conhecidos como máquinas virtuais (VM, *Virtual Machine*) a aplicações virtuais. Este trabalho irá considerar esta

última abordagem, apresentando a implementação da virtualização em um desses níveis de abstração nas próximas subseções. Ainda, o seguinte conceito de virtualização será considerado: “Virtualização é uma tecnologia que combina ou divide recursos computacionais para apresentar um ou muitos ambientes operacionais utilizando metodologias como particionamento ou agregação de hardware e software, simulação completa ou parcial de máquinas, emulação², compartilhamento de tempo e muitos outros” (Chiueh e Brook, 2005). Smith e Nair (2005) chama a atenção para a semelhança entre os conceitos de virtualização e abstração. A diferença é que a segunda tem como objetivo tornar a camada mais abaixo mais simples para os usuários das camadas superiores, enquanto a virtualização não tem este objetivo. A Figura 2.1 mostra esta diferença utilizando como exemplo discos de armazenamento. Em (a) o sistema operacional apresenta o dispositivo aos usuários como arquivos com tamanhos específicos ao invés de trilhas, setores e cabeças, como é enxergado por ele. Já em (b), a camada de virtualização apresenta discos que são acessados pelos usuários desta camada utilizando trilhas, setores e cabeças como em um disco físico.

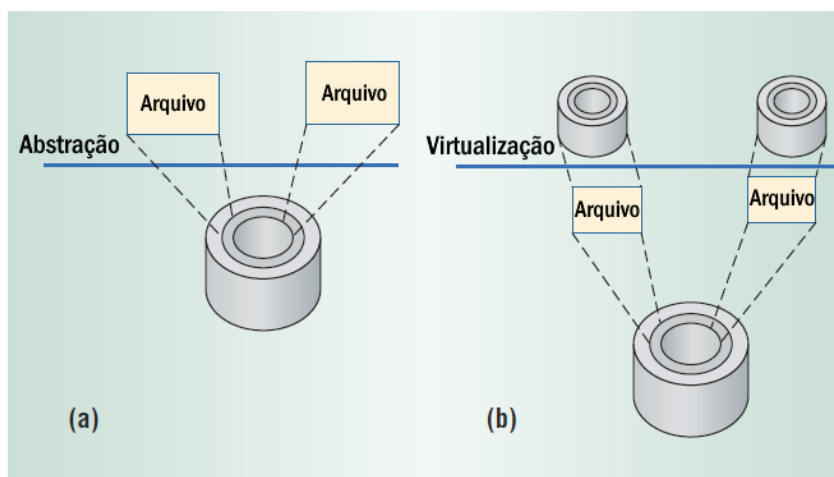


Figura 2.1: Abstração e virtualização aplicadas a discos de armazenamento.

Fonte: Adaptado de (Smith e Nair, 2005)

A Figura 2.2 mostra os níveis de abstração onde é possível implementar a virtualização. A camada de hardware pode ser virtualizada através de duas abordagens distintas (Chiueh e Brook, 2005):

Emulação do *Conjunto de Instruções de Arquitetura (ISA)* É implementada realizando a emulação em software do conjunto de instruções de uma determinada arquitetura. Desta forma, é possível emular todo o comportamento de um computador independente da arquitetura do hardware subjacente. Neste nível, todas as instruções da arquitetura a ser emulada (x86, por exemplo) são traduzidas para o conjunto de instruções do hardware onde está executando (PowerPC, por exemplo). Apesar desta abordagem ser simples, portátil e de fácil implementação, seu desempenho é bastante penalizado

²Emulação é a técnica de interpretar instruções completamente por software (Chiueh e Brook, 2005).

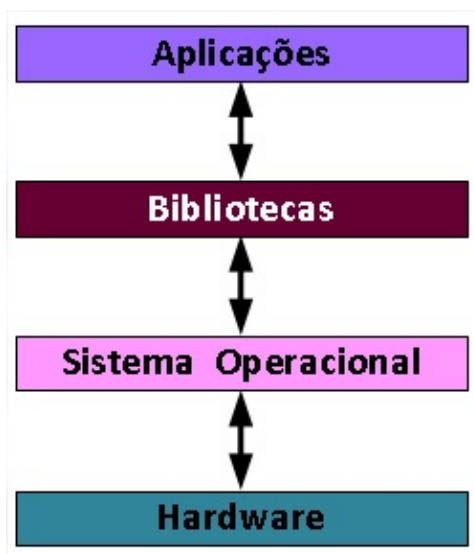


Figura 2.2: Camadas potenciais para implementação da virtualização.
Fonte: Adaptado de (Chiueh e Brook, 2005)

por causa da necessidade de interpretação de todo o conjunto de instruções ser realizado inteiramente em software. Como exemplo desta implementação temos o QEMU (Bellard et al., 2017), um emulador de código aberto que possui dois modos de operação, o modo usuário, onde aplicações desenvolvidas para Linux ou BSD podem ser executadas em diversas arquiteturas emuladas mesmo sendo compiladas para uma específica e o modo de emulação completa, onde um sistema completo incluindo processadores e periféricos é emulado, permitindo que um sistema operacional possa ser executado neste ambiente emulado. O QEMU utiliza um espaço de memória para armazenar instruções já executadas anteriormente e assim diminuir o tempo gasto para fazer a tradução de instruções.

Virtualização da *Camada de Abstração de Hardware (HAL)*³ Diferente da abordagem anterior, esta baseia-se na similaridade entre as arquiteturas do hardware onde o virtualizador está executando e o fornecido pela *Virtual Machine (VM)* ao sistema operacional interno a ela (conhecido como Guest OS ou Sistema Operacional Hóspede). Logo, se o virtualizador executa sobre a plataforma x86, esta mesma plataforma será fornecida pelas instâncias de VMs geradas por este virtualizador. Desta forma ganha-se desempenho, uma vez que as instruções podem ser apenas mapeadas para serem executadas no processador físico sem a necessidade de interpretação ou emulação via software, mas perde-se a possibilidade de executar códigos compilados para plataformas diferentes da hospedeira. A implementação desta abordagem se dá introduzindo uma camada de software chamada Gestor de Máquinas Virtuais (VMM, Virtual Machine Monitor) logo acima do hardware com o objetivo de criar e gerenciar uma ou mais VMs. Esta camada tem a função principal de interceptar as instruções privilegiadas executadas pelo sistema operacional hóspede e realizá-las diretamente no hardware subjacente. O vSphere Hypervisor, produzido pela VMWare (VMWare Inc., 2017), é um exemplo de virtualizador que utiliza esta abordagem. O Xen Hypervisor (Xen Project Community, 2016) também faz

parte desta categoria de virtualizadores e foi o primeiro a ser lançado com código fonte aberto⁴.

A **virtualização implementada na camada de Sistema Operacional** é baseada no compartilhamento tanto do hardware quanto do Sistema Operacional da máquina física, posicionando a camada de virtualização no topo do SO onde uma partição é criada para cada máquina virtual e contém uma cópia do ambiente operacional existente na máquina física para cada partição. As aplicações acessam o hardware através de um conjunto de funções fornecidas pelo sistema operacional conhecidas por Chamadas de Sistema (System Calls). O isolamento e segurança necessários para que as máquinas virtuais possam executar de forma independente são garantidos pela camada de virtualização que fornece Chamadas de Sistema virtuais, garantindo o total controle sobre as operações executadas pelas aplicações. Esta abordagem traz um desempenho muito próximo ao obtido quando se executa as aplicações diretamente no SO do hardware físico, ao custo de ter como limitação o fato de apenas prover ambientes operacionais iguais ao existente na camada subjacente (Chiueh e Brook, 2005; Smith e Nair, 2005). São exemplos de virtualização neste nível o FreeBSD Jail (FreeBSD, 2017) e o projeto Linux Containers (Linux Containers, 2017).

Em geral as chamadas de sistema são acessadas pelas aplicações através de bibliotecas. Estas comumente apresentam API formalmente definidas e que podem ser interceptadas tendo sua execução desviada para um trecho de código específico, técnica conhecida como *hook*. Desta maneira, os programadores podem inserir códigos durante a execução do sistema para alterar o seu comportamento. A virtualização no nível de bibliotecas é implementada interceptando APIs do sistema para controlar a comunicação entre estas e as aplicações. A camada de virtualização então é responsável por apresentar uma API diferente ou uma Interface de Aplicações Binárias (ABI, Application Binary Interface) que disponibilizam acesso ao hardware através da *Conjunto de Instruções de Arquitetura* (ISA) do usuário e da interface de chamadas do sistema (System Call Interface) diferentes da original. Um exemplo muito conhecido é o WINE(WineHQ, 2017), uma camada de virtualização que permite a execução de sistemas compilados para o sistema operacional Windows em outros sistemas operacionais, como o Linux e o Unix.

O último nível de virtualização é que tem um nível mais alto: a aplicação. Sua principal vantagem é a portabilidade entre diferentes arquiteturas, onde os códigos binários podem ser executados sem a necessidade de recompilação. A estratégia para implementação é simples: Cria-se uma máquina virtual que expõe um conjunto de instruções virtual e API próprios. Desta forma, um compilador pode gerar um código abstrato compatível com a ISA virtual especificada pela VM que poderá ser executado em qualquer plataforma que implemente uma máquina virtual capaz de entender e executar este conjunto de instruções. Criar a camada de virtualização é mais simples que desenvolver

⁴A virtualização no nível da *Camada de Abstração de Hardware* (HAL) pode ser do tipo 1 ou “bare metal”, onde o sistema de virtualização é instalado diretamente no hardware como seu sistema operacional ou do Tipo 2, ou “hosted”, instalado sobre um Sistema Operacional já pré-existente, como o Windows e o Linux. O vSphere Hypervisor e o Xen Hypervisor são exemplos de virtualizadores do Tipo 1, já o VirtualBox, da Oracle Inc., e o VMWare Workstation são exemplos de virtualizadores do Tipo 2.

um compilador completo, pois uma VM contém um interpretador para o código binário da aplicação. Em implementações mais sofisticadas, pode-se ainda compilar este código resultado em um código de máquina para execução direta nas camadas subjacentes à de virtualização. As implementações mais conhecidas deste nível de virtualização são o Java Virtual Machine, originalmente criada pela Sun Microsystems que foi posteriormente adquirida pela Oracle Inc., e a Microsoft .NET Common Language Infrastructure (Chiueh e Brook, 2005; Smith e Nair, 2005).

Este trabalho utiliza a virtualização a nível de HAL, portanto, para as demais seções e capítulos será utilizado o termo virtualização como sinônimo para este nível de virtualização.

2.2.1 Arquitetura, Desafios e Soluções

Conforme pôde ser visto nas subseções anteriores, o componente principal da virtualização é a camada de abstração de hardware localizada entre o hardware real e o Sistema Operacional que suporta as aplicações e é conhecida como Monitor de Máquinas Virtuais, (*VMM, Virtual Machine Monitor*). Para Smith e Nair (2005), “uma discussão sobre VMs é também uma discussão sobre arquitetura de computadores”. Esta afirmação faz bastante sentido quando observamos que na arquitetura de virtualização clássica, descrita na Figura 2.3, o Virtual Machine Monitor (VMM) assume o papel do Sistema Operacional que gerencia o equipamento real em um sistema não virtualizado e as máquinas virtuais são como processos ligados à VMM. A VMM é também executada em modo supervisor⁵, tendo então todo o conjunto de instruções do hardware real disponível. Já as máquinas virtuais executam em modo usuário, tendo disponível para si apenas uma parte do conjunto de instruções e, assim como os processos de usuários nos sistemas não virtualizados, executam instruções de hardware e operações de I/O por intermédio do sistema subjacente, neste caso o VMM (Menascé, 2005; Rosenblum e Garfinkel, 2005; Smith e Nair, 2005).

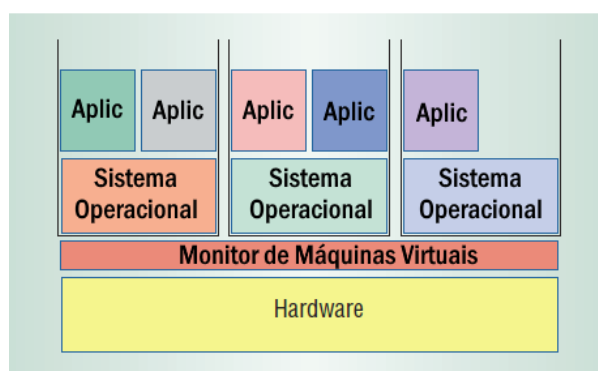


Figura 2.3: Modelo de virtualização clássica.

Fonte: Adaptado de (Rosenblum e Garfinkel, 2005)

Entender quando um determinado hardware pode ser virtualizado passa por entender

⁵Também conhecido como modo núcleo ou modo kernel (Tanenbaum e Filho, 1995)

como os modos usuário e supervisor funcionam e pelo conceito de interrupções em software *TRAPs*, ver (Tanenbaum e Filho, 1995). São definidos dois conjuntos de instruções que executam no modo supervisor, um denominado *instruções sensíveis* composto de instruções que só podem ser executadas no modo supervisor, como instruções de E/S, e outro denominado *instruções privilegiadas*, que é capturada por *TRAPs*. Uma *TRAP* é uma instrução que realiza o chaveamento da execução do modo usuário para o supervisor. Quando um processo no modo usuário tenta executar uma instrução privilegiada é gerada uma *TRAP* para o tratamento desta instrução no modo supervisor. Em um sistema virtualizado, quando uma VM gera uma *TRAP* a VMM verifica se esta foi gerada pelo sistema operacional ou por uma aplicação de usuário e então transfere a execução para o hardware, no primeiro caso, ou então emula o comportamento do hardware e responde à *TRAP* como se esta fosse tratada realmente pelo hardware (Tanenbaum e Filho, 1995; Rosenblum e Garfinkel, 2005). Assim, uma máquina pode ser virtualizada se o conjunto de instruções sensíveis for um subconjunto das instruções privilegiadas (POPEK; GOLDBERG, 1974 apud TANENBAUM; FILHO, 1995). O principal desafio é que o Sistema Operacional Convidado (Guest OS) executa no modo usuário acreditando estar no modo supervisor, contudo, uma característica da plataforma 386 é que, ao contrário dos IBM VM/370, uma parte das *instruções sensíveis* não gera *TRAP* quando executadas no modo usuário, sendo simplesmente ignoradas. Outro problema existente é em relação às instruções que fornecem informações sensíveis ao SO, como o seu modo de execução, que não geram *TRAPs* e não permitem assim que o VMM possa intervir na resposta dada pela CPU, podendo induzir o SO tomar decisões equivocadas. Para contornar estes desafios algumas soluções foram pesquisadas e adotadas, tais como (Tanenbaum e Filho, 1995; Rosenblum e Garfinkel, 2005):

- a) **Tradução binária (Virtualização Total ou Full Virtualization):** Na tradução binária, o VMM examina os blocos de instruções que estão sendo executados em busca de instruções sensíveis não-virtualizáveis, ou seja, que não geram *TRAPs* quando executadas em modo usuário. Para cada instrução encontrada o VMM não a executa diretamente no hardware, mas desvia sua execução para uma rotina interna de tratamento. Uma vez traduzidas estas instruções são guardadas pelo VMM para serem utilizadas em execuções futuras. As demais instruções são executadas diretamente pelo hardware de acordo com seu nível (usuário ou supervisor) mantendo a velocidade que teriam sem a tradução binária.
- b) **Para-virtualização (Virtualização assistida pelo Sistema Operacional):** Nesta abordagem, as chamadas às instruções sensíveis não-virtualizáveis são substituídas por chamadas ao hipervisor conhecidas como hypercalls diretas do SO hóspede a uma API do VMM, trazendo uma maior simplicidade à implementação. Contudo, ao contrário da virtualização total, para realizar esta implementação é necessária a realização de modificações do SO hóspede, pois naturalmente ele não conhece as hypercalls, e aí está o grande ponto negativo da para-virtualização. Apesar de ter um melhor desempenho à medida que elimina sobrecargas da virtualização (*TRAPs*, por exemplo), vários sistemas (como o Windows) não possuem código fonte disponível, dependendo exclusivamente dos desenvolvedores.

- c) **Assistência por hardware:** Esta abordagem remete aos IBM VM/370, trazendo modificações ao hardware que facilitam a sua virtualização, como a adição de um novo modo de execução que permitia a execução direta de instruções provenientes das máquinas virtuais. Alguns processadores, família Intel e AMD, por exemplo, implementaram em seus processadores esta abordagem.

Atualmente a Intel e AMD, principais fabricantes de hardware para a plataforma x86, disponibilizam a cada nova versão atualizações na assistência por hardware que trazem cada vez mais a execução de processos em CPUs virtuais (vCPU) para próximo da execução em CPUs reais. Contudo, os desenvolvedores de sistemas de virtualização associam esta tecnologia com as soluções de tradução binária e para-virtualização, dentre outras, em busca do melhor desempenho para os sistemas que rodam sobre o hipervisor.

2.2.2 Virtualização de Sistemas de Tempo Real

Alguns sistemas necessitam que certas operações executem em um determinado tempo para que o comportamento esperado por ela não seja alterado. Em alguns casos, estas restrições são muito críticas, como nos sistemas para controle automático de voo em aeronaves, onde a informação recebida por sensores é utilizada para a manutenção do voo. Por outro lado, existem sistemas onde os requisitos temporais são mais flexíveis, como em sistemas multimídia onde pode haver alguma degradação no serviço mas que pode ser corrigida em seguida. Esses tipos de sistemas são conhecidos como sistemas de tempo real e precisam de um tratamento diferenciado para que seus requisitos temporais possam ser cumpridos.

Farines et al. (2000), Macêdo et al. (2004) entendem um Sistema de Tempo Real como um caso especial de sistema computacional que tem como principal característica a interação com um ambiente recebendo informações através de dispositivos conhecidos como sensores ou, ainda, através de operadores, e reagindo por meio de atuadores⁶ em prazos específicos que são definidos pelo ambiente que estão interagindo ou ainda por exigências dos serviços oferecidos a um usuário (como no caso dos sistemas multimídia). O que importa na execução de tais sistemas é o conceito de previsibilidade:

Um sistema de tempo real é dito ser previsível (predictable) no domínio lógico e no domínio temporal quando, independentemente de variações ocorrendo no nível de hardware (i.e. desvios do relógio), da carga e de falhas, o comportamento do sistema pode ser antecipado, antes de sua execução (Farines et al., 2000).

Para garantir a previsibilidade de um sistema é preciso levar em consideração o seu comportamento, atentando-se para os piores casos. Além da carga e das falhas, outros aspectos devem ser observados no projeto de um sistema de tempo real, tais como a arquitetura de hardware, sistema operacional e as linguagens de programação envolvidas. Então, a previsibilidade não depende necessariamente do quão rápido se executa uma tarefa, mas sim de questões como quando e em quanto tempo ela será completamente

⁶Um sensor é um dispositivo que reage a estímulos físicos enviando sinais ou dados para outro dispositivo. Um atuador é um mecanismo de potência utilizado para efetuar um movimento ou manter a posição de um robô (Yaskawa America, Inc, 2017). *Tradução livre.*

executada, em cenário de pior caso (Farines et al., 2000). Uma tarefa é “uma sequência de ações executadas pelo sistema, e acionada pela ocorrência de eventos do ambiente”, e é descrita por atributos. Os mais comumente utilizados são (Macêdo et al., 2004):

Deadline (D) Este atributo define o prazo máximo relativo ao momento de ativação da tarefa para que ela seja executada. As implicações práticas em não atender ao *deadline* define o tipo de tarefa: Para tarefas **Críticas ou *hard-realtime***, a perda de um prazo representa perda de vidas humanas, desastres ambientais, destruição de equipamentos ou ainda grandes perdas financeiras. Para tarefas **Firmes ou *firm-realtime***, o resultado após seu *deadline* já não é mais útil. Para tarefas **Brandas ou *soft-realtime***, a perda de um prazo traz degradação do serviço, logo não é desejável, mas suas consequências são consideradas aceitáveis. Neste tipo a resposta após o tempo ainda pode ser aproveitável, ao contrário dos demais.

Período (P) Este atributo tem relação com os momentos em que as tarefas serão ativadas. As tarefas *periódicas* têm uma frequência de ativação em um intervalo regular denominado *Período* que produzem instâncias desta tarefa também conhecidas como “jobs”. Já as tarefas *aperiódicas* não têm uma frequência definida de ativação, podendo acontecer a qualquer momento. Em geral, não se sabe nada sobre quando uma tarefa aperiódica será executada, sendo totalmente aleatória. Quando existe um intervalo mínimo a ser cumprido entre as ativações de uma tarefa aperiódica esta é denominada *esporádica* (Macêdo et al., 2004; Buttazzo, 2011; Farines et al., 2000).

Tempo de Início (S) É o instante em que inicia o processamento de uma tarefa (ativação) (Macêdo et al., 2004).

Jitter (J) Representa “a variação máxima de duas ativações consecutivas que uma tarefa periódica ou esporádica pode apresentar.” Este atributo pode influenciar diretamente no comportamento temporal do sistema (Macêdo et al., 2004; Buttazzo, 2011).

Tempo de Computação (C) É o tempo gasto para uma tarefa executar completamente sem contabilizar as interrupções⁷ (Macêdo et al., 2004; Farines et al., 2000; Buttazzo, 2011).

Um evento pode ocasionar uma reação do Sistema de Tempo Real, desde que seja mapeado como relevante pelos sensores e operadores que fazem parte do ambiente. Estes eventos relevantes são transmitidos ao sistema sob a forma de estímulos (por exemplo, sinais eletrônicos) que dão início ao processamento de uma ou mais tarefas que, por sua vez, pode iniciar outras a depender do resultado da sua execução. Observando os atributos descritos acima e sabendo que o comportamento temporal de uma tarefa T é descrito pela

⁷Uma interrupção na execução da tarefa é conhecida como preempção, podendo ocorrer devido a vários fatores, como a chegada de uma tarefa mais prioritária ou o acesso a um dispositivo entrada e saída (Tanenbaum e Filho, 1995).

quádrupla (J, C, P, D) é possível entender o quanto estão relacionadas e a importância de organizar corretamente a ordem e o momento em que cada *job* (instância de uma tarefa) será executado pelo processador. Para resolver esta questão foram criados algoritmos de escalonamento para atender aos requisitos temporais dos sistemas de tempo-real, sejam eles periódicos ou aperiódicos (Buttazzo, 2011; Macêdo et al., 2004; Farines et al., 2000).

Um sistema de virtualização precisa levar em consideração os requisitos temporais das aplicações hospedadas em suas máquinas virtuais mesmo que estas possuam sistemas operacionais de tempo real instalados nela. Isto ocorre pelo fato destes sistemas operacionais funcionarem sob o pressuposto que eles têm total controle sobre o hardware subjacente a eles, o que não acontece em um sistema virtualizado. Conforme visto na [Subseção 2.2.1](#), o VMM intercepta a execução de instruções privilegiadas pelos SOs das VMs a fim de dar o tratamento correto, de forma que já neste processo é possível identificar um aumento na latência da execução destas instruções, o que pode interferir no Tempo de Computação (C) de uma tarefa. Além do aumento na latência, as CPUs virtuais passam por um segundo processo de escalonamento no hypervisor que não é previsto pelo escalonador do SO hóspede, fazendo com que este possa ter uma impressão de CPU ocupada enquanto efetivamente ela está aguardando para ser processada pelo processador físico. Para atender aos requisitos dos Sistemas de Tempo Real é necessário que o VMM garanta um isolamento espacial e temporal entre as VMs que objetive atender a estes requisitos. Por isolamento espacial entende-se que cada VM deve ter seus próprios espaços de endereçamento para acesso a recursos computacionais compartilhados, de forma que o comprometimento de uma VM, seja por invasão ou falha, não comprometa o cumprimento dos requisitos temporais das demais. Já o isolamento temporal refere-se à capacidade do hypervisor de garantir que o cumprimento dos requisitos de uma VM (por exemplo, o *deadline*) não dependa das demais (Kerstan, 2011).

Os virtualizadores mais conhecidos do mercado são desenvolvidos voltados para aplicações de propósito geral, onde é importante o usuário das aplicações uma resposta rápida, mas não se define formalmente quando esta resposta precisa estar disponível. Até certo ponto a influência de uma máquina virtual em outra é tolerável, embora não desejável. São exemplos destas aplicações sistemas financeiros, serviços de Internet (e-mail, web, etc), dentre outros, sendo que o suporte para VMs de tempo real se dá mais por recomendações (Ex.: Fixar a máquina virtual em um *core* restrição na quantidade de VMs). Contudo, algumas áreas que demandam sistemas de tempo real já podem ter vantagens em utilizar plataformas virtualizadas para simplificar e baratear a implementação das suas aplicações. Kerstan (2011) cita os sistemas embarcados complexos como uma dessas áreas. Ele exemplifica que alguns carros modernos possuem em torno de 70 controladores diferentes que desempenham tarefas distintas variando entre *hard real-time*, *soft real-time* e sem restrições temporais para controlar desde atuadores até dispositivos multimídia interagindo entre si. Neste âmbito, o isolamento espacial e temporal de um sistema virtualizado poderia permitir a diminuição de custos com a substituição de alguns controladores específicos por processadores de uso geral utilizando-os para executarem tanto máquinas virtuais com restrições temporais que demandam funcionalidades específicas de baixo nível como controle de sensores e atuadores quanto as de propósito geral, que demandam funcionalidades de mais alto nível, como APIs para dispositivos multimídia (Kerstan,

2011).

Muitos trabalhos já estão sendo desenvolvidos tanto pela área acadêmica quanto pela indústria para o desenvolvimento de VMMs que atendam adequadamente tanto aos requisitos temporais das máquinas virtuais de tempo real quanto às de propósito geral. [Gu e Zhao \(2012\)](#) apresenta uma vasta pesquisa sobre estes estudos. Seu trabalho aborda os padrões que orientam a construção de camadas de isolamento temporal e espacial como o ARINC 653, destinado ao uso em aeronaves, e o MILS, aderente a aplicações críticas em geral, citando também algumas implementações de virtualizadores baseados nestes padrões. Mais adiante são apresentadas abordagens para sistemas de virtualização, estando elas agrupadas conforme algumas de suas características:

Soluções baseadas nos virtualizadores Xen e KVM. São utilizadas várias abordagens que modificam os escalonadores padrão do Xen e do KVM, assim como a implementação de novos escalonadores e técnicas de balanceamento de carga entre os núcleos levando em consideração quesitos de localidade, como proximidade de cache. Alguns trabalhos também propõem melhorias para acesso a dispositivos de Entrada/Saída.

Soluções baseadas em MicroKernel. São apresentados vários trabalhos baseados em MicroKernel, em especial o L4 e suas variações.

Virtualização de Sistema Operacional. Soluções baseadas em implementações de virtualização em SO, que têm se mostrado mais leves que a virtualização a nível de sistema. São apresentadas algumas soluções comerciais. Algumas implementações voltadas com utilização em telefones móveis também são apresentadas.

Escalonamento granular a nível de tarefas. Também são apresentadas algumas propostas cujo objetivo é fazer com que o VMM passe a ter conhecimento das tarefas com restrições temporais que executam dentro das máquinas virtuais. São descritas tanto propostas que implicam em modificação do SO hóspede através da para-virtualização quanto propostas que não têm este pré-requisito. Neste último caso, o VMM faz análises no comportamento da VM e realiza algumas inferências, alterando então a forma de escalonamento delas.

[Kerstan \(2011\)](#) apresenta em sua tese um VMM híbrido (suporta aplicações para-virtualizadas e totalmente virtualizadas) baseado no processador PowerPC 405 voltado para sistemas embarcados que gerencia tanto máquinas virtuais contendo aplicações de tempo real quanto de propósito geral. O objetivo deste virtualizador é o suporte de aplicações de tempo real, diminuindo, conseqüentemente, os custos e a complexidade na implementação dos sistemas embarcados, atualmente distribuídos em muitos controladores para tarefas específicas.

[Xi et al. \(2011\)](#) apresenta o RT-Xen, um framework para escalonamento hierárquico de tempo real para o hypervisor Xen. O presente trabalho utiliza o RT-Xen como parte integrante da Nuvem de Simulação proposta no Capítulo 4. A [Subseção 2.2.3](#) abordará o RT-Xen com maiores detalhes.

2.2.3 RT-Xen

O RT-Xen, proposto por [Xi et al. \(2011\)](#), é uma extensão do hypervisor de código aberto Xen de modo a utilizar algoritmos de escalonamento de tempo real baseados em um modelo de prioridade fixa ([Farines et al., 2000](#); [Buttazzo, 2011](#)). Em sua proposta, [Xi et al. \(2011\)](#) avalia quatro algoritmos de escalonamento baseados em prioridade fixa⁸ implementados no hypervisor Xen, são eles: Deferrable Server, Sporadic Server, Periodic Server e Polling Server ([Farines et al., 2000](#); [Buttazzo, 2011](#)). Seu estudo conclui: a) Embora algoritmos mais complexos tragam uma maior sobrecarga, a diferença entre eles não é relevante. b) Para tarefas de tempo real brandos, o algoritmo Deferrable Server tem o melhor desempenho em relação aos demais, enquanto que o Periodic Server se mostrou inferior quando o sistema está sobrecarregado. Como fundamentação teórica, o hypervisor Xen será apresentado a seguir e, em seguida, a versão 2.0 do RT-Xen, utilizada na Nuvem proposta no [Capítulo 4](#).

O Xen é um hypervisor de tipo 1⁹ com código fonte aberto licenciado sob a General Public License (GPL) versão 2 e tutorado pela The Linux Foundation ([The Linux Foundation, 2017](#)). Permite a execução simultânea de máquinas virtuais chamadas de *domínios* com hardware virtual paravirtualizados ou completamente virtualizados onde diferentes sistemas operacionais ditos *hóspedes* podem ser instalados. A [Figura 2.4](#) descreve como os domínios que utilizam paravirtualização realizam o acesso a dispositivos de I/O utilizando o domínio privilegiado *Dom0*. Este domínio é conhecido como Domínio de Controle ou ainda Console de Serviço, possui acesso direto ao hardware, abriga os drivers de controle do *hardware*, executa os drivers de *backend*¹⁰ que tratarão as requisições de hardware vindas dos drivers de *frontend*¹⁰ dos demais domínios [Xen Project Community \(2015a\)](#). O escalonador padrão utilizado pelo Xen para executar os demais domínios é o Credit Scheduler ([Xen Project Community, 2017a](#)), um algoritmo de escalonamento para propósito geral proporcional baseado em pesos. Para cada domínio é estipulado um peso e um limitador (cap). O peso indica a relação de prioridade entre os domínios, enquanto o limitador estipula o consumo máximo de CPU que um domínio poderá consumir. Estes domínios são executados pelo VMM sem diferenciação entre eles e sem qualquer conhecimento acerca das restrições que as tarefas executantes no SO hóspede possuem. A partir do Xen 4.5 o RT-Xen 2.0 foi adicionado ao código do Xen, possibilitando que os requisitos de tempo real definidos para as VMs sejam considerados durante o escalonamento das suas processadores virtuais (vCPUs), introduzindo uma priorização diferenciada para estes domínios e permitindo que tarefas com restrições de tempo brandas possam ser executadas em sistemas com processadores multi-núcleo¹¹ mantendo uma boa eficiência.

Para o encaminhamento do tráfego de rede de/para as máquinas virtuais a literatura oficial do Xen descreve três topologias que podem ser configuradas ([Xen Project](#)

⁸Nestes algoritmos a prioridade das tarefas não se modifica, pois são definidas no projeto do escalonador.

⁹Instalado diretamente no hardware como seu sistema operacional.

¹⁰Para esta monografia, os termos *frontend* e *backend* se referem às camadas que representam, respectivamente, a origem de uma requisição e o seu destino.

¹¹Sistemas multi-núcleo ou multi-core abrigam mais de um núcleo em um único processador, permitindo que alguns níveis de cache existentes neste processador sejam compartilhados entre os núcleos.



Figura 2.4: Acesso a dispositivos em domínios paravirtualizados.
 Fonte: Traduzido de Wiki Xen Project (Xen Project Community, 2015b).

Community, 2017b):

- Baseada em roteamento: Cada interface virtual dos domínios não privilegiados (DomU) é ligada à interface de *backend* do domínio dom0. Desta forma, os pacotes vindos destes domínios são roteados de forma simples baseado na tabela de roteamento¹² do sistema operacional.
- Baseada em pontes: Uma ponte (mais conhecida pelo termo *bridge*) é configurada no sistema operacional do domínio de controle e conectada ao dispositivo de rede de *backend* (Ver Figura 2.4) deste domínio. Opcionalmente, a interface de rede real pode ser também adicionada a esta configuração. A configuração utilizando uma ponte permite que os domínios virtuais acessem a rede como se fossem dispositivos reais.
- Baseada no Open vSwitch (The Linux Foundation, 2016): Assemelha-se à topologia baseada em pontes, porém as interfaces virtuais dos domínios são conectadas ao Open vSwitch, um switch virtual de código aberto mantido pelo projeto *The Linux Foundation* (The Linux Foundation, 2017).

Xi et al. (2014) apresenta um estudo empírico do framework RT-Xen 2.0 que é uma evolução do trabalho realizado em 2011, sendo implementado e avaliado um esquema de escalonamento baseado em três dimensões:

Escalonamento. São avaliados dois escalonadores: *rt-global* (escalonamento global) e *rt-partition* (escalonamento particionado). Um escalonador particionado escalona as vCPUs na fila de execução do seu respectivo núcleo enquanto um escalonador global escalona as vCPUs em uma fila única que atende a todos os núcleos (Xi et al., 2014).

¹²A tabela de roteamento contém todos os caminhos possíveis de um pacote ser encaminhado para fora do ambiente do sistema operacional.

Prioridades. As prioridades de cada domínio podem ser definidas estaticamente ou dinamicamente para cada escalonador através dos algoritmos *Earliest Deadline First* (EDF) ou *Deadline Monotonic* (DM) (ver (Farines et al., 2000)).

Servidores. As vCPUs são implementadas como os servidores Deferrable Server ou Periodic Server. Em (Xi et al., 2011) é possível ver a descrição, implementação e avaliação destes servidores além do Sporadic Server e Polling Server na plataforma Xen.

Xi et al. (2014) apresenta uma experimentação que além de atestar o melhor desempenho dos escalonadores propostos em relação ao Credit Scheduler, leva em consideração o escalonador utilizado no SO hóspede. As avaliações realizadas atestam a capacidade do RT-Xen para escalonar tarefas de tempo real brandas em sistemas mono e multi-core permitindo a flexibilidade na escolha da política de escalonamento desejada. A execução simultânea de máquinas virtuais com restrições temporais e de propósito geral também é uma característica importante deste trabalho que interessa, sobretudo, a área de sistemas embarcados. Estas também são características interessantes para a Nuvem proposta no presente trabalho e descrita no Capítulo 4.

2.3 NUVEM ROBÓTICA

A Organização Internacional para Padronização (ISO, International Organization for Standardization) define um robô como “um mecanismo atuado (dotado de atuadores) programável em dois ou mais eixos¹³ com algum grau de autonomia¹⁴, que se move em seu ambiente para realizar tarefas planejadas (ISO, 2012)”.¹⁵ O International Organization for Standardization (ISO) ainda define duas classes de robôs que se diferenciam pela sua utilização:

- a) Robôs industriais: São robôs multifuncionais controlados automaticamente e programáveis em no mínimo três eixos que podem ser fixos ou móveis. São reprogramáveis e utilizados especificamente na área industrial.
- b) Robôs de serviço: São robôs utilizados para realizar tarefas não industriais, geralmente para algum ser humano.

Muitas tarefas já são realizadas por robôs, sejam por: a) serem muito simples e tediosas, como limpeza doméstica e alimentação de animais; b) estarem além dos limites físicos dos seres humanos, como a exploração em ambientes inóspitos e cirurgias laparoscópicas; c) são muito perigosas, como o desarmamento de artefatos explosivos; ou d) por serem executados com menor custo e/ou maior eficiência, como a montagem de carros.

¹³Um eixo é “uma direção usada para especificar o movimento do robô em um modo linear ou rotativo” (ISO, 2012) - *Tradução livre*.

¹⁴A autonomia é “a habilidade de realizar tarefas planejadas baseado no estado atual e em detecção sensorial sem intervenção humana (ISO, 2012) - *Tradução livre*.

¹⁵Tradução livre.

Com o crescimento da sua utilização para a realização de tarefas realizadas por humanos a composição dos robôs tem se tornado mais complexa, exigindo novos atuadores e sensores, bem como novos algoritmos responsáveis por elementos funcionais como reconhecimento de voz e imagem, navegação, dentre outros (Kamei et al., 2012). Um robô é limitado pela sua capacidade computacional, de armazenamento e energética existentes em seus circuitos, assim como na programação que ali está embarcada. Tarefas mais complexas exigem algoritmos que utilizam mais processamento, memória e espaço de armazenamento para o registro de dados oriundos dos sensores. Pode ser necessário também a utilização de um conjunto de sensores que não possa ser instalado em um único robô, seja por restrições de espaço físico ou alto custo para a implementação de um sistema que coordene todos os tipos diferentes de sensores. Integrar os robôs em rede foi uma solução dada para estes desafios (Hu et al., 2012).

O IEEE (2015) define os Robôs em Rede (Networked Robots) como “um dispositivo robótico¹⁶ conectado a uma rede de comunicação como a Internet ou uma Rede Local. A rede pode ser ou não cabeada e baseada em qualquer protocolo de rede como TCP, UDP ou 802.11¹⁷”. Este comitê ainda define duas subclasses de robôs em rede: a) teleoperados, onde “um humano envia comandos e recebe respostas do robô via rede”; b) autônomos, “onde os robôs e sensores trocam dados através da rede, ampliando o campo de percepção dos robôs”.¹⁸ O avanço das tecnologias de comunicação em rede foi um motivador para a integração dos robôs a fim de estender sua capacidade computacional (Agostinho et al., 2011). Quando esta rede é formada por mais de um robô, ainda é possível distribuir entre os participantes da rede o trabalho de sensoriamento, atuação, comunicação e processamento, compartilhando informações de modo que uma tarefa mais ampla possa ser realizada de através da cooperação entre os robôs da rede (Hu et al., 2012).

Contudo, a rede robótica apesar de ter uma capacidade computacional e de armazenamento estendida é composta de um conjunto limitado de componentes, que por sua vez tem um limite destes recursos (e outros, como capacidade energética, tamanho etc.) que restringem os trabalhos executados pela rede robótica (Hu et al., 2012). Tarefas como navegação e mapeamento de ambientes dinâmicos, troca de mensagens para manutenção de rotas entre os robôs da rede, entre outras, exigem um alto processamento. A Computação em Nuvem ajuda a transpor estes limites fornecendo recursos computacionais de forma elástica através de sua infraestrutura amplamente acessível para que as redes robóticas possam utilizá-la como um “cérebro remoto”, processando cargas de trabalho pesadas remotamente e recebendo seu resultado. Grandes volumes de informações sobre o ambiente podem ser armazenadas e organizadas na nuvem, estando assim acessível para que os robôs da rede possam aprender com a história uns dos outros. (Hu et al., 2012).

Em 2010, James Kuffner, da Google, propôs a utilização da Internet como um recurso

¹⁶Um dispositivo robótico é “um mecanismo atuado (dotado de atuadores) que atende às características de um robô industrial ou de serviço, mas que não possui um número de eixos programáveis ou um grau de autonomia” (ISO, 2012). Tradução Livre.

¹⁷O padrão 802.11 é definido pelo IEEE, disponível em (<http://standards.ieee.org/about/get/802/802.11.html>).

¹⁸Tradução Livre.

disponível para uso da rede robótica, podendo esta se beneficiar da capacidade de processamento em paralelo e o compartilhamento de dados que a Nuvem¹⁹ oferece. Neste trabalho, ele introduziu o termo “Cloud Robotics” (Kuffner, 2010), ou nuvem robótica. Como exemplo desta abordagem, há o projeto de carro autônomo da Google, que indexa mapas em imagens coletadas através de satélites, do Google Streetview e contribuições de usuários através da rede para tornar a localização mais exata (Goldberg e Kehoe, 2013). Já o projeto RoboEarth (RoboEarth Project, 2014) objetiva a utilização da Internet para a criação de um grande banco de dados em rede aberto que estará disponível para ser acessado e atualizado por robôs ao redor do mundo. A utilização desta base visa diminuir os limites em relação ao entendimento de ambientes não estruturados que os robôs têm quando precisam realizar uma nova tarefa. Tendo acesso aos dados fornecidos por outros robôs, é possível entender estes ambientes sem a necessidade de uma total reconstrução do seu modelo de trabalho a cada nova tarefa a ser executada (RoboEarth Project, 2014). Estes dois exemplos dão uma ampla noção do poder que a computação em nuvem pode fornecer às redes robóticas. Mais do que isto, a computação em nuvem permite que processadores mais baratos possam ser incorporados a uma grande gama de dispositivos, estreitando assim a relação entre os conceitos de Nuvem Robótica, Automação e Internet das Coisas (Goldberg e Kehoe, 2013). Segundo James Kuffner, a nuvem pode tornar os robôs mais simples, baratos e inteligentes (Guizzo, 2011).

Um outro motivador para a Nuvem Robótica é a mobilidade. Com o avanço das pesquisas no campo da robótica, os robôs e sistemas autônomos passaram a se movimentar por uma gama maior de terrenos e por um tempo maior. A Boston Dynamics (Guizzo, 2011), recém adquirida pela Google, desenvolve robôs que conseguem se movimentar em campos abertos, matas, montanhas, escombros, paredes verticais, dentre outros. Aeronaves não tripuladas já possuem uma grande autonomia de voo, sendo utilizados em missões de reconhecimento territorial varrendo grandes áreas. Dentro desta realidade, a distribuição geográfica existente nas Nuvens Computacionais é um fator que possibilita com que as informações necessárias para um robô conectado em rede estejam fisicamente mais próximas, diminuindo o tempo para estas bases serem acessadas ou atualizadas. Além do menor tempo de acesso às informações, o grande número de processadores existentes nos Centros de Dados (Datacenters) que compõem as nuvens computacionais permite que a rede robótica tenha a disposição um grande poder de processamento paralelo de algoritmos mais sofisticados (Lorencik e Sincak, 2013).

2.3.1 Características

A ideia de robôs estarem conectados à nuvem é muito recente. Por este motivo, as definições e características da nuvem robótica ainda não são um consenso entre os autores. Contudo, grandes contribuições têm sido dadas neste sentido, sendo possível identificar algumas como sendo importantes características deste novo campo de pesquisa.

¹⁹Este termo também é utilizado como uma metáfora para Internet

2.3.1.1 Independência de Hardware Poder desenvolver as aplicações para Rede Robótica independente do hardware utilizado nos dispositivos é uma característica chave para o progresso da Nuvem Robótica (Kamei et al., 2012). Esta independência permite:

- a) O desenvolvimento de aplicações para a Rede Robótica por desenvolvedores que não possuem um conhecimento aprofundado do hardware para o qual estão desenvolvendo (Kamei et al., 2012);
- b) Reutilização de aplicações para diferentes plataformas robóticas (Du et al., 2011; Chen et al., 2010);
- c) Aumento do tempo de vida dos serviços robóticos individuais (sensores, atuadores, etc), mesmo no caso do rápido desenvolvimento de novas tecnologias (Kamei et al., 2012).

2.3.1.2 Comunicação em Rede Robótica A capacidade de comunicação em rede é um pré-requisito básico para um robô ou sistema autônomo estar ligado à Nuvem. Hu et al. (2012) resalta a importância da comunicação em rede entre robôs como forma de expandir sua capacidade computacional e/ou permitir sua tele operação e monitoramento a distância. Seu trabalho posiciona a Rede Robótica como um passo evolutivo para a Nuvem Robótica e define uma arquitetura para esta nuvem baseada na utilização conjunta de dois níveis de comunicação: máquina-para-máquina (M2M, Machine-to-Machine) e máquina-para-nuvem (M2C, Machine-to-Cloud). O primeiro nível é a conexão entre dispositivos, que formam uma rede Robótica, enquanto o segundo é a comunicação entre os dispositivos da Rede Robótica e a Nuvem Computacional (Figura 2.5). Agostinho et al. (2011) utiliza a plataforma REALabs para facilitar o desenvolvimento de aplicações para Redes Robóticas e descreve a integração com um ambiente de Computação em Nuvem baseado em tecnologias de virtualização de servidores.

2.3.1.3 Web Services Agostinho et al. (2011), Du et al. (2011) mostram a importância da utilização de Web Services na implementação de plataformas robóticas para a nuvem. Já Koubâa (2014) foca a utilização de Web Services para a virtualização de dispositivos. Em seu estudo, ele cria uma camada de virtualização de dispositivos que abstrai o software e hardware como serviços web. A utilização de Web Services nas Nuvens Robóticas é herdada da Computação em Nuvem, que já a utiliza na entrega de serviços sob demanda (Koubâa, 2014).

2.3.1.4 Compartilhamento de conhecimentos Uma das principais características das Nuvens Robóticas é a colaboração entre dispositivos para compartilhamento de conhecimento e troca de informações semânticas. Mais do que isto, a Nuvem provê uma grande massa de dados que podem ser adquiridas pelos robôs para executarem tarefas. A disponibilidade de bases de dados contendo conhecimentos e comportamentos já prontos para utilização diminui o tempo e custo do aprendizado de máquina ao passo que maximiza a utilização dos robôs, uma vez que eles podem aprender com o histórico de

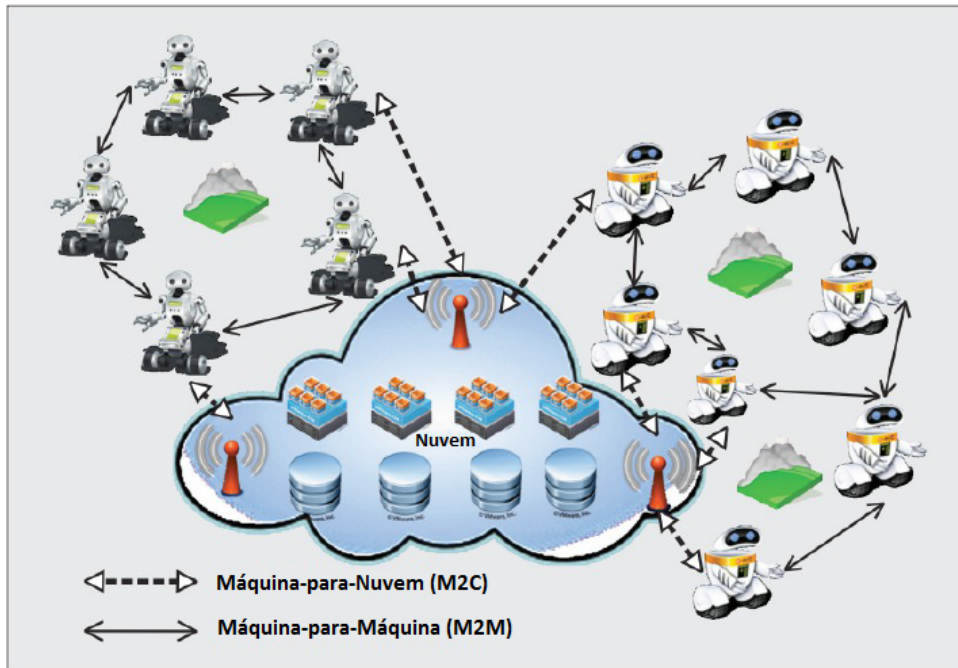


Figura 2.5: Comunicação M2C e M2M.

Fonte: Hu et al. (2012)

outros robôs da nuvem e realizarem tarefas diferentes das originalmente programadas ou atuarem em novos ambientes (Goldberg e Kehoe, 2013; Yates et al., 2011; Hu et al., 2012; Kamei et al., 2012).

2.3.1.5 Execução de processamento e armazenamento offboard Absorver o processamento de tarefas que fazem uma utilização intensiva de CPU e/ou o armazenamento de dados que não estão em uso no momento também está no conjunto das características mais importantes da Nuvem Robótica. Além disto, as aplicações poderão utilizar hardware especializados para tarefas específicas, como GPUs para processamento matemático (Goldberg e Kehoe, 2013; Agostinho et al., 2011; Kuffner, 2010; Koubâa, 2014; Hu et al., 2012; Kamei et al., 2012).

2.3.2 Desafios

2.3.2.1 Escalonamento de Tarefas Conforme já foi visto anteriormente conectar-se a uma Nuvem Robótica estende consideravelmente o campo de atuação de uma Rede Robótica principalmente quando tarefas são executadas na Nuvem. Contudo, para que a Rede Robótica obtenha efetivamente esta vantagem, é necessário escalonar as tarefas a serem executadas sob dois aspectos principais:

a) Onde executar a tarefa?

Fazer a correta escolha das tarefas que serão enviadas para a nuvem é um quesito

importante de otimização dos recursos. Segundo [Hu et al. \(2012\)](#), esta decisão deve levar em consideração três estratégias de execução:

- Execução da tarefa realizada localmente no robô;
- Execução da tarefa realizada de forma colaborativa pelo grupo de robôs da rede robótica;
- Execução na Nuvem.

Segundo [Hu et al. \(2012\)](#), uma estratégia possível para realizar esta decisão é, por exemplo, objetivar a minimização do consumo de energia do robô dentro dos limites de tempo de execução para a tarefa. Para isto, propõe comparar a energia gasta na execução local da tarefa com a energia a ser gasta transmitindo-a à Nuvem para execução.

b) Qual a ordem de execução das tarefas submetidas à Nuvem?

Ordenar a execução das tarefas submetidas a uma Nuvem Robótica de forma a atender tanto aos requisitos funcionais quanto aos de Qualidade de Serviço (QoS) é um desafio importante para a otimização da Nuvem Robótica e deve levar em consideração o custo e as restrições de tempo para a execução da tarefa. [Du et al. \(2011\)](#) constrói um escalonador para sua proposta de nuvem robótica, dita Robot Cloud Center, que se baseia nestes quesitos. Também é mostrado em seu trabalho a simulação que comprova a eficácia deste escalonador. Já [Agostinho et al. \(2011\)](#) propõe um processo onde o agrupamento de serviços que realizam funções especializadas é realizado utilizando a técnica de “dividir para conquistar”.

2.3.2.2 Atraso na comunicação Um dos aspectos importantes tanto para a Nuvem Robótica quanto para a Rede Robótica é a comunicação. Seja entre máquinas ou entre máquinas e nuvem, decisões acerca da execução das tarefas devem ser tomadas levando em consideração o atraso existente nesta comunicação. Conforme [Hu et al. \(2012\)](#), transferir a execução de uma tarefa para a nuvem (ou outros robôs membros da rede) só vale a pena se o custo energético de transferência for menor do que o de execução local e se esta transferência não exceder os limites máximos de execução estipulados para a tarefa. Em outras palavras, se o atraso existente na comunicação for baixo o suficiente transferir a tarefa em tempo hábil e esta transferência for capaz de ser realizada deixando ainda energia suficiente para o robô completar a tarefa, valerá a pena executar a tarefa remotamente. [Goldberg e Kehoe \(2013\)](#) ressalta que enquanto algumas aplicações não são ainda aderentes à execução na nuvem, como por exemplo tarefas com requisitos de tempo real, muitas outras já o são, como análise de vídeo e imagens online.

2.3.2.3 Aprendizado de Máquina Tornar o aprendizado de máquina um processo rápido e eficaz é talvez um dos desafios que pode trazer os robôs cada vez mais próximos do homem. Complementar o aprendizado dos robôs com conhecimentos que a alta capacidade cognitiva do ser humano pode gerar, como a experiência e intuição, pode ser

aproveitado para resolver uma grande quantidade de problemas como a rotulação de imagens para visão computacional (Goldberg e Kehoe, 2013).

2.3.2.4 Segurança Segurança é um dos desafios naturais das Nuvens por estas serem um ambiente naturalmente compartilhado, pois ser multi-hospedeiro é uma característica da Nuvem. Dentro do ambiente de uma Nuvem Robótica segurança também é uma preocupação. Alguns dos desafios de segurança que precisam ser endereçados são:

- a) Acesso não autorizado aos robôs através das aplicações em execução nas VMs por causa de uma baixa proteção na utilização de recursos (Agostinho et al., 2011);
- b) Garantir a confiabilidade da infraestrutura de virtualização a fim de evitar que VMs maliciosas sejam inseridas no ambiente e possam causar danos ao sabotar tarefas importantes (Hu et al., 2012);
- c) Permitir que os robôs possam verificar a confiabilidade das Nuvens Públicas antes de delegarem tarefas a estas (Hu et al., 2012);
- d) Garantir a integridade, segurança e confiabilidade do armazenamento de informações pertencentes às aplicações (Hu et al., 2012).

2.4 SISTEMA OPERACIONAL ROBÓTICO

O Sistema Operacional Robótico (ROS, Robot Operating System) é um framework de código aberto, com ênfase na pesquisa integrativa de robótica em larga escala, que adiciona uma camada de comunicação estruturada aos sistemas operacionais dos diferentes nós de um cluster computacional heterogêneo (Quigley et al., 2009). Baseada nos desafios de pesquisa do campo da robótica, a arquitetura do ROS teve em seu projeto cinco objetivos principais com os quais alcançou aplicação em uma gama abrangente de sistemas robóticos (Quigley et al., 2009):

Rede Ponto-a-Ponto (Peer-to-Peer). Os processos que compõem um sistema desenvolvido no ROS comumente estão distribuídos em computadores diferentes e podem estar conectados entre si por uma rede heterogênea, que se beneficia da topologia ponto-a-ponto para a troca de mensagens entre estes processos principalmente quando envolve redes mais lentas, como as redes sem fio.

Multi-linguagem. O ROS possui um sistema de mensagens independente da linguagem de programação que permite o uso simultâneo de diversas linguagens no desenvolvimento de uma aplicação. Para isto, a especificação do ROS é realizada no nível do gerenciamento das mensagens que são trocadas entre nós e serviços, não chegando camadas mais baixas evitando as especificidades de uma linguagem específica. O suporte ao desenvolvimento entre linguagens é alcançado utilizando-se para a troca de mensagens uma interface de definição de linguagem (IDL, Interface Definition Language) simples baseada em pequenos arquivos de textos que descrevem os campos de cada mensagem.

Baseado em ferramentas. Para melhor gerenciar a complexidade do ROS seus desenvolvedores optaram por um projeto baseado em um micronúcleo (microkernel) com um conjunto de ferramentas auxiliares onde cada uma realiza tarefas específicas como navegar na árvore de códigos fonte, alterar parâmetros de configuração, dentre outros. Segundo a avaliação dos desenvolvedores os ganhos obtidos com estabilidade e gerenciamento da complexidade superam as perdas de eficiência.

Leve. Os desenvolvedores do ROS orientam que drivers e algoritmos sejam implementados em bibliotecas sem qualquer dependência do ROS. Desta forma, os executáveis criados são pequenos, pois se limitam a expor as funcionalidades da biblioteca para o ROS, facilitando também o re-uso de código e a realização de testes unitários.

Código fonte aberto e livre. O código fonte do ROS está disponível publicamente sob os termos da licença BSD, que permite o desenvolvimento de aplicações tanto comerciais como não-comerciais. O ROS transfere dados entre módulos utilizando comunicações entre processos (IPC), não sendo necessário que os módulos estejam ligados a um mesmo executável, permitindo assim que módulos com licenças diferentes possam fazer parte de um mesmo software.

Para entender o funcionamento básico do ROS os conceitos dos elementos nós, mensagens, tópicos e serviços. A [Figura 2.6](#) mostra como estes elementos se relacionam para formar o esquema de comunicação do ROS. *Nós são processos que realizam alguma tarefa computacional.* Um sistema geralmente é composto de vários nós que se comunicam através de **mensagens**. Uma mensagem é *uma estrutura de dados fortemente tipada* onde são suportados tipos primitivos padrões (ex.: inteiros, booleanos, etc), coleções (arrays) destes tipos, outras mensagens e coleções de mensagens. Para um nó enviar uma mensagem ele a **publica** em um **tópico**, que é um nome simples que identifica o frame relacionado a mensagens publicadas. Uma mensagem é repassada para todos os nós que se **inscreveram** naquele tópico. Mais de um nó pode publicar ou se inscrever em um ou mais tópicos, não havendo, ainda, nenhuma relação entre os nós, pois eles sabem da existência um do outro. O último conceito básico do ROS é o de **serviço**, que traz uma comunicação entre nós apropriada para transações síncronas, o que não ocorre com o esquema de "broadcast" utilizados pelos tópicos. Um serviço é *definido por um nome e um par de mensagens fortemente tipadas: uma para a requisição e uma para a resposta.* *Ao contrário dos tópicos, apenas um nó pode publicar em um serviço.* (Quigley et al., 2009).

O desenvolvimento de software com o ROS tem algumas características bastante atraativas não apenas para pesquisas acadêmicas, mas também para o uso comercial. Para o presente estudo, algumas são importantes para o entendimento da arquitetura de nuvem robótica proposta no [Capítulo 4](#):

Desenvolvimento e detecção de erros. A arquitetura distribuída baseada em nós existente no ROS permite que suas comunicações possam ser desenhadas baseadas em grafos, sendo que um grafo pode ser modificado dinamicamente sem a necessidade de reiniciar todos os nós, mas apenas os afetados pela mudança. Isto é

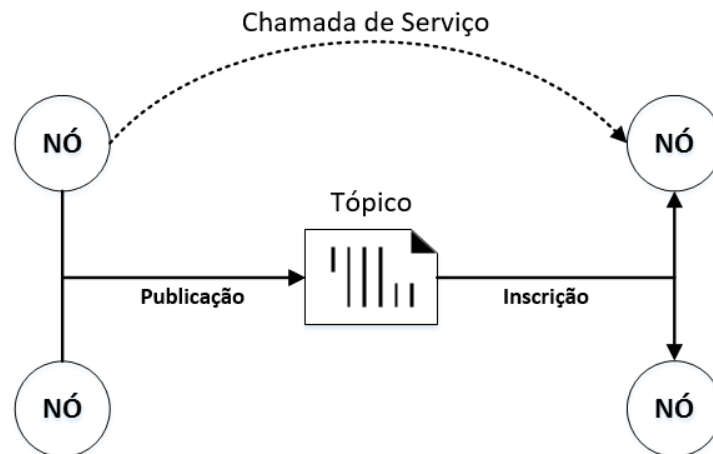


Figura 2.6: Relação entre os elementos que compõem o ROS.
 Fonte: Baseado em <http://wiki.ros.org/ROS/Concepts>.

particularmente vantajoso na busca por erros ou pontos de otimização do sistema, pois os nós podem ser adicionados ou removidos individualmente com operações simples de inicialização e parada do processo, respectivamente. Outro caso de uso relevante é a capacidade de captura e armazenamento de dados oriundos dos sensores e mensagens trocadas entre os nós, podendo ser reproduzidos posteriormente no mesmo grafo ou em um outro sem que haja qualquer modificação no código fonte dos nós envolvidos. Na prática, isto permite que, por exemplo, as imagens capturadas de uma câmera possam ser utilizadas como entrada em um grafo onde um nó que realiza a detecção de objetos esteja em desenvolvimento (Quigley et al., 2009).

Empacotamento de módulos e distribuição colaborativa. Para facilitar a execução de operações como inicialização e parada dos processos que fazem parte de uma funcionalidades, o ROS disponibiliza a ferramenta *roslaunch*, que realiza a inicialização dos processos descritos em formato XML (W3C, 2008), e persistidos em um arquivo que é lido pelo *roslaunch*. Para permitir o desenvolvimento colaborativo o ROS é organizado em pacotes, onde um pacote é um diretório contendo um arquivo no formato XML que descreve aquele pacote e indica suas dependência. Um repositório ROS é então organizado como uma árvore de diretórios contendo pacotes em suas folhas (Quigley et al., 2009).

Composição de funcionalidades. Anteriormente, foi visto que a ferramenta *roslaunch* é capaz de fazer a instanciação automática de um conjunto de nós baseada em uma descrição em arquivo XML, o que não atende à necessidade de uma determinada funcionalidade ser instanciada mais de uma vez, como por exemplo em um sistema multi-robôs onde a funcionalidade de navegação deve ser instanciada para cada robô. O ROS permite que os nós e arquivos de descrição do *roslaunch* sejam movidos para espaços de nomes (namespaces) filhos a fim de garantir que não hajam colisões de nome. Uma funcionalidade específica é chamada de pilha, ou *stack*, e é

definida como *um conjunto de nós que realiza algo útil*, por exemplo, o sistema de navegação citado anteriormente (Quigley et al., 2009).

Esta seção descreveu os conceitos e características básicas do framework ROS, apresentou a arquitetura distribuída baseada em troca de mensagens e alguns casos de aplicação. Os pontos abordados são importantes para a compreensão do simulador apresentado na Seção 3.3, que compõe a camada de simulação da arquitetura proposta no Capítulo 4 junto com o simulador robótico gazebo, apresentado na Seção 2.5.

2.5 GAZEBO

O Gazebo é um simulador dinâmico 3D com a capacidade de simular de forma precisa e eficiente populações de robôs em ambientes internos e externos complexos (OSRF, 2014b). Os objetos simulados se comportam de forma realística, pois possuem aspectos físicos como massa, velocidade, fricção, dentre outros atributos. Os robôs são compostos de corpos rígidos conectados via articulações ou junções conhecidas como *joints* que se movimentam de acordo com as forças (angulares ou lineares) aplicadas nestas superfícies e junções. A simulação dos sensores que podem ser adicionados aos objetos fornecem uma resposta realística do ambiente que, em conjunto com a proximidade existente entre o comportamento do hardware simulado e o real, permite que os ambientes dinâmicos simulados possam ser utilizados no processo de desenvolvimento dos sistemas robóticos (Koenig e Howard, 2004).

A Figura 2.7 faz a descrição da arquitetura do Gazebo. Esta arquitetura permite a criação de novos robôs, atuadores, sensores e outros objetos facilmente devido ao que é chamado de *modelos* (models), uma API simples para a adição destes objetos. Um *mundo*, ou *world*, é uma coleção de modelos e sensores. Um modelo pode ser definido como *qualquer objeto que mantém uma representação física*, sendo formado por pelo menos um *corpo*, ou *body*, qualquer quantidade de sensores e junções, e interfaces. Um corpo é um bloco básico de construção de um modelo, sendo representado fisicamente por formas geométricas. Cada corpo possui aspectos físicos como massa e atrito, assim como propriedades visuais, como cor e textura, podendo ser conectado a outro através de junções de tipos variados, como juntas deslizantes ou dobradiças, para formarem relações cinemáticas e dinâmicas. As juntas ainda podem atuar como motores, pois quando uma força é aplicada a uma delas o atrito o corpo conectado e outros resulta em movimento. Os sensores atuam na coleta de dados ou, caso seja um sensor ativo, no envio de dados. No Gazebo, ele é um dispositivo abstrato que só ganha um “corpo” quando adicionado a um modelo. São exemplos de sensores odômetro, câmeras, sonares, dentre outros (Koenig e Howard, 2004).

Dois importantes componentes do Gazebo são o motor de física e a biblioteca gráfica. Estes dois componentes localizam-se na camada mais baixa da arquitetura. O motor de física é o responsável por simular os aspectos dinâmicos e cinéticos referentes aos corpos articulados descritos nos modelos. Para facilitar o suporte e representação de múltiplos motores físicos, uma camada de abstração foi implementada logo acima desta, assim é possível escolher entre diversos motores disponíveis. A biblioteca gráfica é responsável por gerar as formas gráficas em 3D definidas nos modelos e fornecer a interface que apresentará

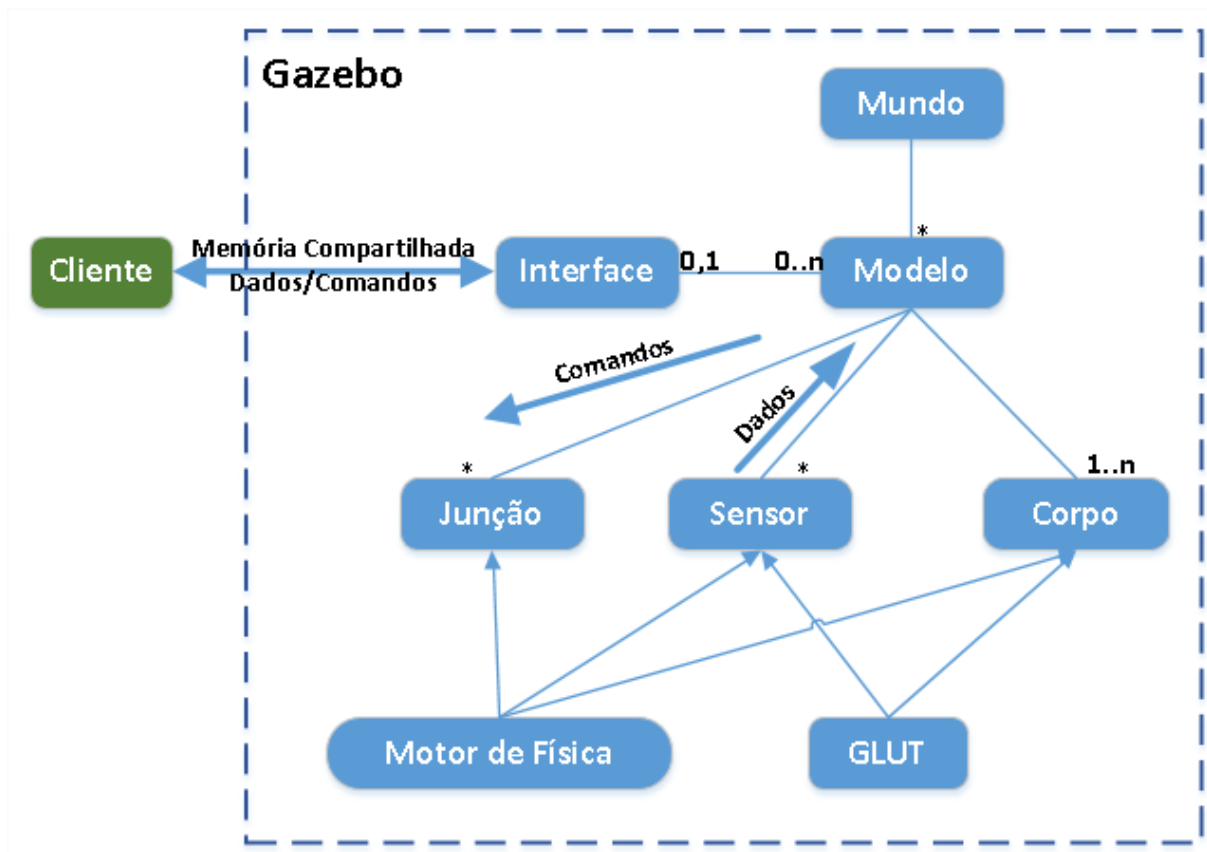


Figura 2.7: Estrutura geral do Gazebo.
 Fonte: Baseado em Koenig e Howard (2004)

ao usuário o ambiente simulado. Originalmente, foi utilizado a biblioteca OpenGL e o GLUT, um conjunto de ferramentas para o gerenciamento de janelas baseado no OpenGL. Atualmente são utilizados o motor gráfico OGRE e a biblioteca QT, respectivamente (Koenig e Howard, 2004).

A interação do cliente com os modelos do ambiente é realizada através de interfaces que permitem a definição de velocidades, a leitura de dados dos sensores e imagens das câmeras, e inserir objetos ao ambiente em tempo de execução. Originalmente, isto era feito através da manipulação de uma memória compartilhada entre o cliente e a interface, mas nas versões atuais um novo componente, o *Gazebo Master*, tem a função de fazer o gerenciamento de tópicos, fornecendo assim uma estrutura de comunicação distribuída semelhante ao utilizado pelo ROS (Koenig e Howard, 2004).

Esta seção descreveu os conceitos e características básicas do gazebo e apresentou sua arquitetura. Os pontos abordados são importantes para a compreensão do simulador apresentado na Seção 3.3, que compõe a camada de simulação da arquitetura proposta no Capítulo 4.

2.6 PLATAFORMAS REALABS E REALCLOUD

REALabs é uma plataforma distribuída para rede robótica proposta por (Cardozo et al., 2010). Ela adiciona uma camada acima dos frameworks robóticos a fim de permitir o acesso remoto a estes utilizando protocolos, serviços e soluções de segurança abertos e comuns a dispositivos de comunicação modernos, como os *smartphones*. Além de favorecer a comunicação entre domínios, esta proposta também enfrenta os seguintes restrições no acesso a plataformas robóticas em rede (Cardozo et al., 2010):

Protocolos de rede especializados. Os frameworks para robôs móveis utilizam protocolos especializados acessíveis através de APIs desenvolvidas especificamente para eles. Dispositivos móveis modernos como *smartphones* e *tablets* em geral não possuem suporte nativo a estas APIs.

Acesso à rede. Uma topologia de rede comumente adotada é a que utiliza endereços de protocolo Internet (IP, Internet Protocol) privados e filtros de pacotes (Firewalls). A comunicação destes endereços com outras redes ocorre através de funções especializadas como Tradução de Endereços de Rede (NAT, Network Address Translation) e encaminhadores HTTP (IETF, 1999) (Proxies). Esta topologia comumente não permite que os frameworks robóticos acessem ou sejam acessados por uma rede externa quando utilizando IPs privados. Ainda, os filtros de pacotes geralmente bloqueiam acesso a portas que não estejam explicitamente permitidas e há casos onde a alteração das regras deste equipamento não seja possível.

Segurança no acesso aos frameworks. Muitos frameworks tratam questões de segurança como mecanismos para verificação de acesso dos usuários (autenticação e autorização) e a segurança da comunicação, podendo comprometer os recursos robóticos.

Para lidar com estes desafios (Cardozo et al., 2010) propôs a arquitetura mostrada na Figura 2.8. A Interface do Usuário (Front-end) possui componentes que fornecem APIs para que a interação entre as aplicações robóticas e os robôs seja realizada. Estes componentes são instalados na estação de trabalho do usuário. O sistema embarcado é composto por micros servidores capazes de executar no hardware do robô com a finalidade de processar as requisições e realizar uma atuação no robô. O Manipulador de Protocolos é responsável por inspecionar as requisições de recursos (robôs, câmeras, etc), realizar a verificação de segurança e encaminhar a mensagem para os micros servidores embarcados no recurso. Por fim, o Gerenciamento é formado por componentes que realizam ações a nível de federações, estabelecendo relações de confiança com membros de outras entidades da federação, gerenciando usuários, credenciais e certificados, dentre outras funções típicas do Gerenciamento. Os componentes de Gerenciamento e do Manipulador de Protocolos são instalados em servidores que fazem parte da rede onde os robôs estão conectados.

O trabalho de Cardozo et al. (2010) permitiu que desenvolvedores pudessem executar aplicações locais utilizando robôs localizados em laboratórios remotos. Para aplicações que realizam baixa transferência de dados e pouca taxa de processamento este modelo

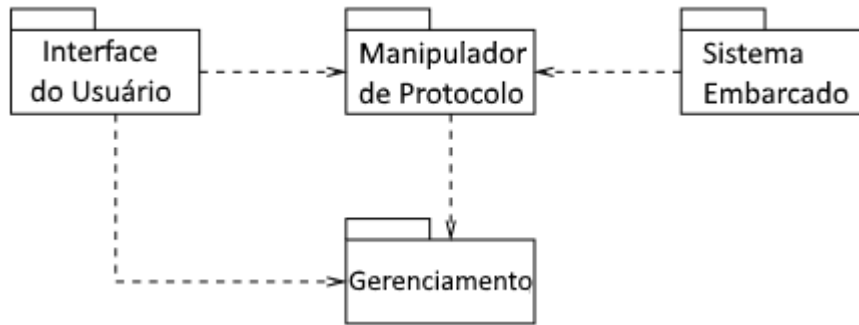


Figura 2.8: Arquitetura da plataforma REALabs.

Fonte: Baseado em (Cardozo et al., 2010).

tem resultados adequados. Contudo, alguns gargalos causam degradação em aplicações que necessitam de comunicação eficiente e alto poder de processamento (Agostinho et al., 2011):

Conexões de internet lentas. Os atrasos introduzidos por links de baixa velocidade associados às inspeções realizadas nas requisições do protocolo HTTP causa degradação no desempenho das aplicações robóticas.

Baixo poder de processamento da estação de trabalho do usuário. Para aplicações que exigem um alto poder de processamento como algoritmos de visão computacional e inteligência computacional o processador utilizado na estação de trabalho do usuário pode não ser suficiente para entregar os resultados no tempo desejado.

Para enfrentar estes desafios, Agostinho et al. (2011) propõe uma plataforma de Computação em Nuvem para oferecer o REALabs como serviço em uma infraestrutura de Nuvem Privada apresentada na figura Figura 2.9. Nesta abordagem, as aplicações robóticas executam na Interface do Usuário contida em uma máquina virtual gerenciada pelo Virtualizador instalado em servidores que fazem parte da mesma rede ou em redes próximas ao REALabs e, conseqüentemente, dos robôs. Esta topologia elimina os atrasos impostos pelas conexões via internet e, no mesmo tempo, oferece ao desenvolvedor o poder de processamento que as plataformas de servidores multiprocessados e multinúcleo são capazes de fornecer. A arquitetura da plataforma ainda prevê a utilização de uma máquina virtual que contém o Gerenciador de VMs, responsável por gerenciar o ciclo de vida das máquinas virtuais no ambiente e configuração do Filtro de Pacotes quando uma VM é criada, e o Validador de Sessão, responsável por atribuir privilégios às VMs que pertencentes a usuários com sessões válidas para acesso. O Validador de Sessão garante a mesma proteção fornecida pelo Manipulador de Protocolo, não sendo então necessário que as requisições oriundas do REALcloud sejam inspecionadas, diminuindo o atraso na comunicação via rede (Agostinho et al., 2011).

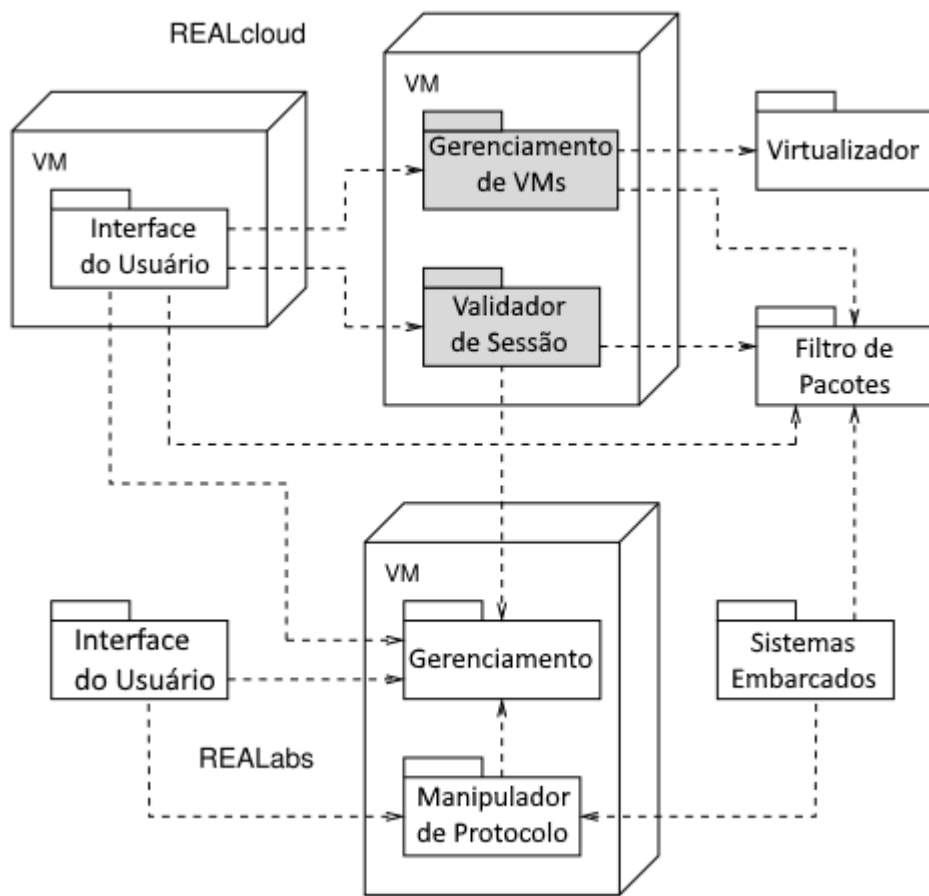


Figura 2.9: Arquitetura da plataforma REALabs.
 Fonte: Baseado em (Agostinho et al., 2011).

QUADROTOR AR.DRONE

3.1 INTRODUÇÃO

Quadrotor é um tipo de helicóptero composto por quatro rotores montados nas extremidades de uma estrutura fixa fina e leve, mas resistente, disposta em forma de cruz. Cada rotor possui um motor com velocidade controlada eletronicamente e ligado através de engrenagens a uma hélice fixa, sendo então um conjunto fixo cuja única variação é a velocidade angular das hélices, que é controlada de forma individual, ou seja, cada hélice pode ter uma velocidade diferente, definindo assim os movimentos realizados pela aeronave. Este modelo agrega ao quadrotor uma boa navegabilidade e a capacidade de pouso e decolagem horizontal, mais conhecida como Decolagem e Aterrissagem Vertical (VTOL, Vertical Take-Off and Landing) (Bresciani, 2008). A Figura 3.1 mostra em (a) que as hélices formam dois pares, cada um com rotores diametralmente opostos, que possuem sentidos de rotação inversos, ou seja, um par gira no sentido horário e outro no sentido anti-horário. Em (b) é possível ver que as hélices geram uma corrente de ar que sopra para baixo (em verde) e, conseqüentemente, uma força vertical que empurra a aeronave para cima (em vermelho). Algumas características fizeram com que o interesse dos pesquisadores nos quadrotores aumentasse bastante nos últimos anos, como seu sistema mecânico simplificado pelo uso de rotores fixos e o controle veicular baseado na variação de suas velocidades é mais simples que o complexo sistema mecânico dos helicópteros. O uso de quatro rotores permite que estes tenham um menor diâmetro que o do rotor principal de um helicóptero, reduz o acúmulo de energia cinética em cada motor, diminuindo o risco de danos caso algum objeto entre em contato com eles durante o voo. O tamanho reduzido dos rotores também torna possível a instalação de proteções ao seu redor, habilitando assim o uso em ambientes fechados ou com muitos obstáculos, uma facilidade para a realização de testes e pesquisas. Todas as vantagens observadas neste tipo de aeronave permitiram que ela fosse utilizada para diversos fins, como vigilância, busca e resgate, redes de sensores móveis, dentre outros (Hoffmann et al., 2007).

O AR.Drone é um quadrotor cujo desenvolvimento iniciou em 2004 pela Parrot com o propósito de atender ao mercado dos jogos eletrônicos e entretenimento pessoal, sendo

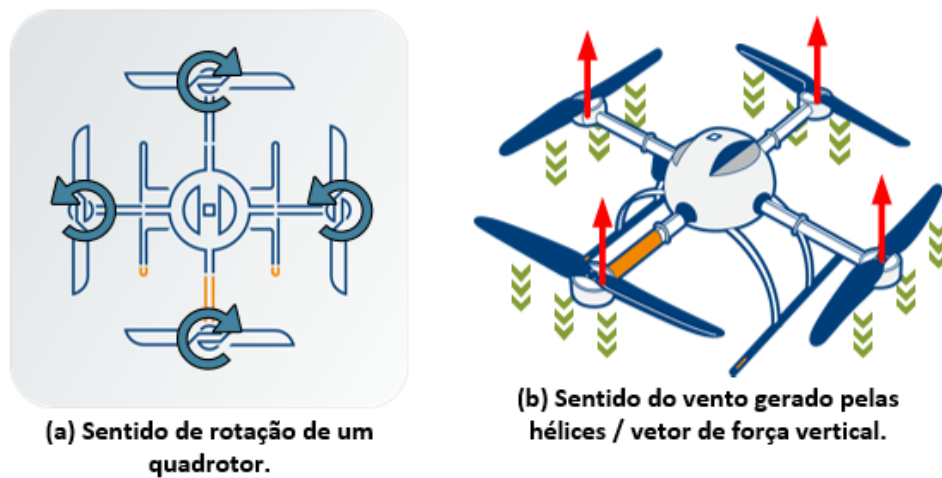


Figura 3.1: Composição de um quadrotor.
 Fonte: Baseado em Piskorski et al. (2012)

lançado ao mercado na Consumer Electronics Show do ano de 2010. Para alcançar este objetivo a Parrot implementou em seu equipamento um conjunto de sensores de baixo custo que, aliados a um modelo aerodinâmico, algoritmos de navegação e processamento de vídeo, garantem voos estáveis (Bristeau et al., 2011). Sua interface de controle inicial foi a linha de equipamentos mobile da Apple (iPhone e iPod Touch). Contudo, com a disponibilização do SDK do AR.Drone pela Parrot e a proposta de uma plataforma aberta para desenvolvedores de jogos logo surgiram plataformas de controle para outros dispositivos, como os smartphones equipados com sistema operacional Android e computadores com sistema operacional Linux (Parrot Inc, 2010).

Os principais aspectos referentes ao funcionamento do AR.Drone foram estudados em fontes variadas, como códigos de terceiros, documentos oficiais da empresa fabricante do equipamento, registros de patentes, estudos acadêmicos e observações do seu ambiente operacional. Desta pesquisa surge um layout das camadas principais de hardware e software onde é possível identificar com clareza a relação entre estas camadas e a descrição não somente desta relação mas de pontos que relacionam estas camadas com o comportamento do drone.

3.2 COMPONENTES DO AR.DRONE

A partir da análise do ambiente operacional do AR.Drone e do estudo de diversos trabalhos dos autores citados neste capítulo, o diagrama da Figura 3.2 foi elaborado demonstrando as camadas de hardware e software presentes no AR.Drone e o relacionamento entre estas camadas.

O Sistema de Propulsão engloba os rotores e o circuito que realiza seu controle sendo descrito na Subseção 3.2.2. O Módulo de Navegação é formado pelos sensores que coletam os dados utilizados para a navegação da aeronave e o microcontrolador responsável por converter o sinal analógico dos sensores em sinais digitais. O Sistema de Visão é formado

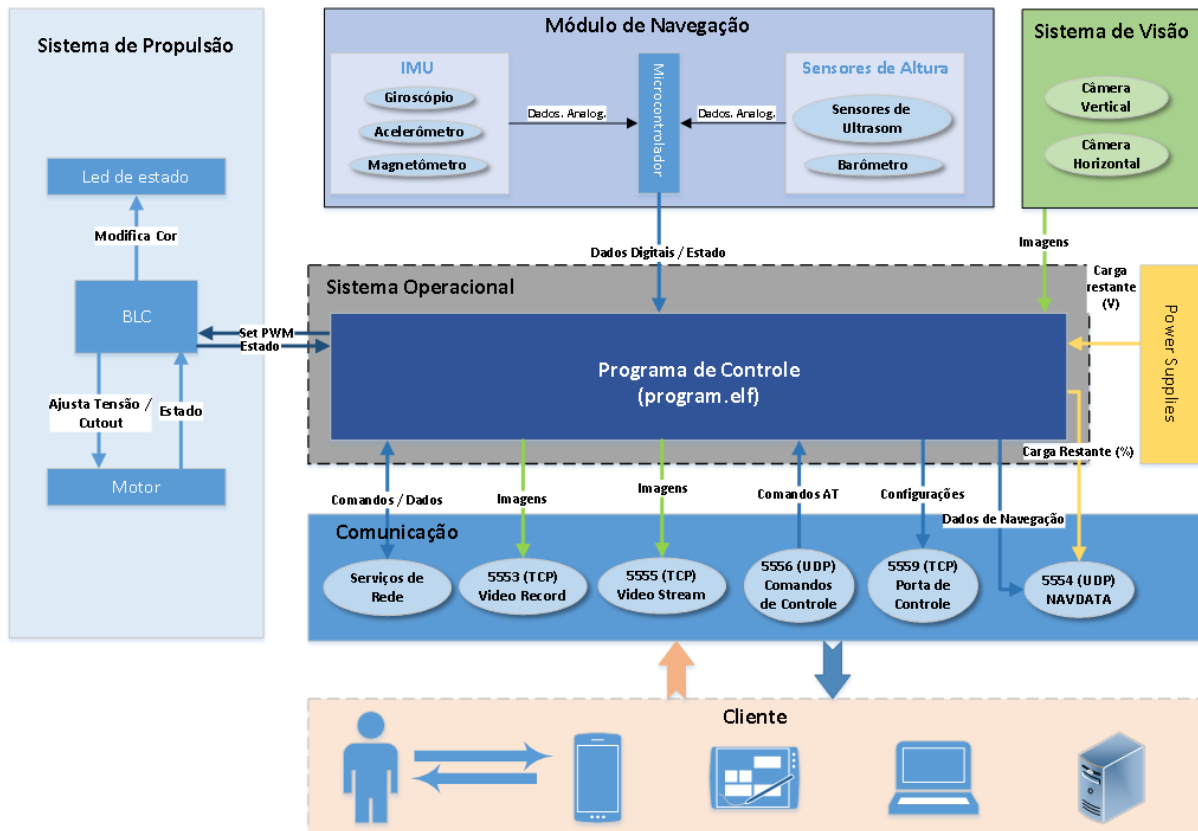


Figura 3.2: AR.Drone - Camadas de hardware e software.

pelas câmeras horizontais e verticais, cujas imagens geradas são utilizadas para a estimativa de velocidade utilizada no sistema de navegação e transmitidas para o usuário. O Sistema de Visão e o Módulo de Navegação serão abordados na [Subseção 3.2.4](#). A Camada de Controle é composta do Sistema Operacional e os sistemas que nele estão embarcados e serão descritos na [Subseção 3.2.5](#). O mais importante dentre estes sistemas é o Controlador Central (CC), descrito na [Subseção 3.2.1](#), pois este realiza todas as tarefas de controle do Drone. [Bristeau et al. \(2011\)](#) descreve os algoritmos de navegação e controle implementados no Controlador Central (CC). A Camada de Comunicação descrita na [Subseção 3.2.6](#) responsável pela interação com o usuário que se conecta a esta camada através da rede sem fio a partir da Camada Cliente a ser descrita na [Subseção 3.2.7](#). Toda a discussão desenvolvida nesta seção é baseada nos AR.Drone versões 1.0 e 2.0.

3.2.1 O Controlador Central (CC)

O termo Controlador Central (CC, Central Controller) é utilizado nos registros de patente submetidos pela Parrot para as tecnologias utilizadas em seus drones e também será adotado neste trabalho. É nesta central que estão implementados os algoritmos que controlam ([Chaperon e Pierre, 2011](#); [Bergner e Regelungstechnik, 2012](#)):

- a) Sistema de propulsão – Transforma as ações de voo ordenadas pelo usuário em

sinais de acionamento para os motores;

- b) Atualização do software de controle dos microcontroladores do sistema de propulsão;
- c) Sistema de comunicação – Controla a comunicação entre a aeronave e o dispositivo de controle do usuário (tipicamente um smartphone ou tablet);
- d) Sistema de navegação – Monitora os diversos sensores instalados no equipamento e comanda ações para estabilização e voo.
- e) Processamento de imagem – Processa as imagens enviadas pelas câmeras para a definição de velocidades horizontal e vertical da aeronave;
- f) Controle de pouso e decolagem - Automatiza os procedimentos de pouso e decolagem da aeronave, facilitando a execução destas ações pelo usuário;
- g) Ações de segurança – Monitora estados de partes do sistema como motores, bateria e canais de comunicação com o usuário ativando medidas de segurança quando necessário.

Para a realização destas tarefas o núcleo (kernel) do sistema operacional de tempo real gerencia threads inicializadas pelo CC (Bristeau et al., 2011): loop de controle principal do AR.Drone, executando a uma taxa fixa de 200 Hz, gerenciamento da comunicação sem fio; coleta de dados das câmeras de vídeo; compressão de vídeo para transmissão sem fio; processamento de imagem; aquisição de dados de navegação (navdata); e estimativa de orientação (attitude) da aeronave. A inicialização das threads pode ser acompanhada através das informações emitidas durante a execução da CC.

A Figura 3.2 reflete a importância do CC como componente central da inteligência embarcada no AR.Drone, pois este se relaciona com todas as camadas realizando tarefas diversas como a coleta de informações do módulo de navegação, monitoramento e envio de comandos para controle do sistema de propulsão, aquisição e manipulação das imagens do Sistema de Visão, monitoramento do nível de carga da bateria, envio informações para o cliente e recebimento dos comandos de controle enviados por este através da Camada de Comunicação. Estas tarefas serão melhor descritas nas próximas subseções.

3.2.2 Sistema de propulsão

O sistema de propulsão do AR.Drone é constituído de quatro unidades de propulsão que são controladas pelo CC. Cada unidade é composta de (Federal Communications Commission, 2010; PaparazziUAV, 2016; Bergner e Regelungstechnik, 2012; Chaperon e Pierre, 2011):

- a) Um motor sem escovas de 14,5 W, alimentado por 12V que alcança uma taxa de rotação de 28.500 rotações por minuto (RPMs);
- b) Uma placa controladora para os motores (Brushless Control Board (BLC)) contendo um micro-controlador de 8 bits responsável pelo o controle e monitoramento do motor;

- c) Um diodo emissor de luz (LED) indicativo de funcionamento do motor, que é controlado também pela BLC;
- d) Uma hélice composta de fibra de carbono.

A inicialização dos motores é feita aplicando-se uma velocidade mínima de 20% do valor máximo seja definida para os microcontroladores, sendo necessário que uma tarefa envie para os rotores periodicamente uma mensagem atualizando sua velocidade de operação. A fim de evitar alterações indesejadas no comportamento do drone a execução dos comandos nos rotores é feita de forma simultânea pelo CC que envia uma única mensagem informando o valor de configuração de velocidade para os motores através de conexões seriais assíncronas existentes entre o CC e cada microcontrolador (Chaperon e Pierre, 2011). O CC também realiza periodicamente a manutenção do estado e cor dos LEDs indicativos de estado do motor.

Outra tarefa realizada pelo CC é a verificação do sistema de proteção conhecido como "Cutout System", implementado na BLC, que monitora os giros do motor e identifica quando este não está operando na velocidade especificada, indicando possíveis falhas como a colisão de uma hélice com um obstáculo, a ausência da hélice no motor, dentre outros. Para evitar que o drone opere com a falta, mal funcionamento ou falha de algum dos motores, uma tarefa é executada periodicamente pelo CC para verificar o estado dos motores e realizar o desligamento de todos quando é detectada a falha em um deles (Chaperon e Pierre, 2011; Bergner e Regelungstechnik, 2012).

A Tabela 3.1 descreve as tarefas principais executadas pelo CC para o controle do sistema de propulsão com suas respectivas restrições de tempo para execução (período, tipo e criticidade) e quais atuadores controla. Estes valores foram extraídos dos trabalhos de (Perquin, 2012) e (PaparazziUAV, 2016).

Tabela 3.1: Tarefas do sistema de propulsão.

Tarefa	Período	Sensor/Atuador	Tipo	Criticidade
Atualizar PWM dos motores	5 milisegundo (ms)	Motores	Periódica	Hard
Testar motores	100 a 125 ms	Motores	Periódica	Hard
Configurar Leds	20 ms	Leds	Periódica	Soft

3.2.3 Verificação de estado da bateria

O AR.Drone possui um mecanismo que realiza o monitoramento da carga da bateria para detectar quando o nível está crítico. A CC é responsável por periodicamente coletar o nível atual da bateria e disponibilizar para o CC. Baseado no nível de carga da bateria dois alertas podem ser emitidos pelo CC (Piskorski et al., 2012; Krajník et al., 2011):

- a) BATTERY LOW ALERT – A carga da bateria está muito baixa para permitir uma nova decolagem. O drone continua funcionando normalmente até que pouse.

- b) BATTERY LOW EMERGENCY – A carga da bateria é baixa demais para garantir a segurança do drone em voo. O procedimento de pouso é iniciado imediatamente.

Os códigos fonte dos CC analisados utilizam um período de 100ms para as verificações do estado de carga da bateria, cuja carga restante em volts (V) é obtida via (Inter-Integrated Circuit), um barramento de conexão bidirecional serial para conectar dispositivos de baixa velocidade como microcontroladores, controladores de E/S, dentre outros periféricos. O valor medido é convertido para percentual pelo processo de verificação de bateria e disponibilizado para o cliente através da porta de dados de navegação.

As tarefas principais para o monitoramento da bateria podem ser vistas na [Tabela 3.2](#) com suas respectivas restrições de tempo para execução (período, tipo e criticidade), quais atuadores controla ou mede e o responsável pela execução da tarefa.

Tabela 3.2: Tarefas do sistema de monitoramento da bateria.

Tarefa	Executada por	Período	Sensor/Atuador	Tipo	Criticidade
Checar carga da bateria	Programa de Controle	100 ms	Bateria	Periódica	Soft
Disponibilizar nível da bateria	placa de navegação	Desconhecido	Bateria	Periódica	Soft

3.2.4 Sistema de Navegação

O AR.Drone possui um módulo de navegação externo à placa mãe que acomoda os sensores de navegação e controle de altura que compõem o equipamento. Sua Unidade de Medição Inercial (IMU) possui um baixo custo e é composta de ([Bristeau et al., 2011](#); [Krajník et al., 2011](#); [Piskorski et al., 2012](#)):

- a) Um giroscópio de 2 eixos (AR.Drone v1);
- b) Um giroscópio de 1 eixos vertical (AR.Drone v1);
- c) Um giroscópio de 3 eixos (AR.Drone v2);
- d) Um acelerômetro de 3 eixos;
- e) Um magnetômetro de 3 eixos;

Para a estimativa de altura e distância do solo são usados:

- a) Dois sensores ultrasônicos utilizados para estimar a altitude e a profundidade das imagens capturadas pela câmera vertical. Este sensor não é capaz de estimar alturas maiores que 6 metros;
- b) Um sensor de pressão (versão 2.0) que possibilita realizar a estimativa de altitude em qualquer altura.

Tabela 3.3: Tarefa principal do sistema de navegação.

Tarefa	Executada por	Período	Atuador	Tipo	Criticidade
Ler dados da navboard	Programa de Controle	5 ms	Navboard	Periódica	Hard

Um microcontrolador PIC de 16 bits é responsável por fazer a leitura periódica dos sensores, realizar a conversão Analógico/Digital destes dados e disponibilizá-los para a placa-mãe com uma periodicidade de 5 ms (Bristeau et al., 2011).

A estimativa de velocidade horizontal em relação ao solo é feita pelo sistema de visão utilizando a imagem capturada pela câmera vertical. Este método é utilizado devido ao alto nível de ruído que os acelerômetros de baixo custo geram e influenciam fortemente na estabilização da aeronave principalmente quando é necessário pairar sobre um ponto específico (manobra conhecida como 'hovering'). Os algoritmos utilizados neste sistema não fazem parte do escopo deste estudo e estão descritos em Bristeau et al. (2011), Derbanne (2013).

A tarefa com tempo identificado na literatura disponível para a navegação é a coleta dos dados de navegação que pode ser vista na Tabela 3.3.

3.2.5 Sistemas Embarcados

O AR.Drone utiliza o kernel Linux nas versões 2.6.27 (AR.Drone 1.0) e 2.6.32 (AR.Drone 2.0), ambos compilados para a plataforma ARM. Sobre este kernel é utilizada a suíte BusyBox, um software onde um único executável combina versões mais enxutas de utilitários comumente utilizados cujo desenvolvimento objetivou atender principalmente sistemas com restrições de recursos, como os sistemas embarcados (Vlasenko, 2017). Analisando o código fonte do kernel é possível observar modificações realizadas pela Parrot para ajustá-lo às necessidades da arquitetura do drone, bem como controladores de dispositivos (drivers) específicos para os dispositivos embarcados. O sistema não possui muitos processos, podendo ser contabilizados em torno de 60 processos executando logo após sua inicialização, incluindo:

- a) Processos ligados ao núcleo do sistema operacional;
- b) Processos responsáveis por serviços de rede;
- c) Scripts de monitoramento de espaço em disco, atualização de firmware e restauração para configuração de fábrica;
- d) proxy de comandos shell, utilizado pelo programa de controle para executar comandos no shell;
- e) Programa de Controle (program.elf), responsável por todos os demais algoritmos de controle do drone.

3.2.6 Comunicação com o cliente

O AR.Drone realiza a comunicação Cliente-Drone utilizando uma rede sem fio pública que é disponibilizada na inicialização do equipamento (Krajník et al., 2011). Nesta rede, são disponibilizados serviços que realizam variadas ações como recepção de comandos, publicação de imagens das câmeras, publicação de dados e navegação, dentre outros. Cada serviço utiliza uma porta de rede, chamada pela documentação da API do AR.Drone, de canal de comunicação. As principais portas são (Piskorski et al., 2012; Pleban et al., 2014):

- a) Transferência de arquivos (21 - TCP¹): Serviço de transferência de arquivos disponível através do protocolo FTP² para permitir que o cliente acesse fotografias e vídeos gerados pelo drone.
- b) Telnet³ (23 - TCP): Serviço de terminal para acesso ao sistema operacional com nível de administrador através do usuário 'root'.
- c) Atualização do firmware (5551 TCP): Utilizado para a o envio de arquivos de atualização do sistema para o drone. Utiliza o protocolo FTP para a realização da transferência de arquivos.
- d) Gravação de vídeo (5553 TCP): Este canal existe a partir da versão 2 do AR.Drone e é utilizado quando a aplicação deseja fazer uma gravação de vídeo. Disponibiliza as imagens da câmera frontal em sua resolução máxima.
- e) Dados de navegação e bateria (5554 UDP⁴): Disponibiliza informações diversas sobre o drone, tais como status, posição, velocidade dos motores, velocidade horizontal do drone, dentre outros. Podem ser configuradas duas periodicidades para a disponibilização destes dados: a cada 66,7ms ou 5 ms, sendo a primeira utilizada no chamado "modo demo", que é o funcionamento padrão do Drone, e a segunda no modo de depuração ou "modo debug", comumente utilizado para identificação de erros. Nesta porta também é disponibilizada a carga restante da bateria. Este valor é fornecido pelo sistema de fornecimento de energia (Power Supplies) em Volts, mas fornecido ao cliente em porcentagem.
- f) Transmissão de vídeo (5555 TCP/UDP): Utilizado para a transmissão contínua de vídeo para o cliente. Na versão 1 do AR.Drone é utilizado o protocolo UDP, já na versão 2 é utilizado o TCP. É possível alternar entre a imagem da câmera frontal e vertical.

¹O protocolo TCP é definido no RFC 793, disponível em <https://tools.ietf.org/html/rfc793>.

²O protocolo FTP é definido no RFC 959, disponível em <https://tools.ietf.org/html/rfc959>

³O protocolo Telnet é definido no RFC 854, disponível em <https://tools.ietf.org/html/rfc854>.

⁴O protocolo UDP é definido no RFC 768, disponível em <https://www.ietf.org/rfc/rfc768.txt>.

- g) Comandos de Controle (5556 UDP): O drone é controlado por comandos AT⁵ emitidos periodicamente para este canal. É esperada uma atualização a cada 33 ms e, caso isto não ocorra em até 50 ms, o drone entrará no estado 'Com Watchdog triggered' e começará a planar sobre um ponto fixo até que receba o comando *AT*COMWDG*. Caso isto não ocorra em até 2000 ms, a comunicação com o cliente é fechada e uma nova conexão precisa ser estabelecida para que o controle da aeronave possa ser re-estabelecido.
- h) Porta de Controle (5559 TCP): Uma comunicação com este canal pode ser estabelecido para troca de informações críticas para o sistema. É comumente utilizada para obtenção da atual configuração do drone ou para a definição de novos parâmetros de configuração.

Uma forte crítica realizada pela comunidade em relação à comunicação do AR.Drone é a fraca segurança que ela tem. A rede sem fio utilizada é pública, sem nenhum tipo de autenticação e criptografia. Os serviços de Telnet e FTP não possuem autenticação e dão acesso a partes importantes do sistema. Pleban et al. (2014) aborda esta insegurança e sugere métodos para adicionar criptografia e autenticação nesta rede sem fio, bem como aponta possíveis ataques.

3.2.7 Cliente

O cliente do AR.Drone tem a função de traduzir as orientações informadas pelo usuário em seu painel de controle em comandos pré-definidos na camada de comunicação do equipamento e realizar o envio destes comandos para o canal mais adequado para tratar desta solicitação. Inclui-se neste processo a aquisição de informações como dados de navegação e imagens. Embora lançado inicialmente para utilização com clientes para dispositivos móveis da Apple como iPhone, iPad e iPod, o AR.Drone possui uma SDK aberta que permite o desenvolvimento de clientes para outras plataformas. Esta SDK permitiu não apenas a expansão para outras plataformas mas também a utilização do AR.Drone em pesquisas científicas na área de navegação autônoma, identificação de objetos, dentre outros (Parrot Inc, 2010; Piskorski et al., 2012; Krajiník et al., 2011). Um passo importante para o desenvolvimento destas pesquisas foi a criação de um controlador de dispositivo para o conjunto de ferramentas para desenvolvimento de aplicações robóticas ROS, descrito na Seção 2.4. Este controlador é também utilizado no ambiente de simulação proposto para o presente trabalho.

3.3 TUM SIMULATOR - O SIMULADOR DO AR.DRONE

O simulador do quadrotor AR.Drone foi desenvolvido por Hongrong Huang e Juergen Sturm do Grupo de Visão Computacional da Technical University of Munich. Distribuído como um pacote do ambiente ROS, denominado *tum_simulator*, este simulador

⁵Os comandos AT foram definidos pela *Hayes Microcomputer Products* como uma linguagem para controle de dispositivos para comunicação e se tornaram um padrão utilizado até os dias atuais. Mais informações disponíveis em: <http://nemesi.lonestar.org/reference/telecom/modems/index.html>.

utiliza, como base para a simulação do AR.Drone, o Hector Quadrotor Stack (Meyer e Kohlbrecher, 2014), um conjunto de plugins cujo objetivo é a simulação de quadrotores com um comportamento próximo ao dos equipamentos reais utilizando o framework ROS e o simulador gazebo. No presente trabalho, este simulador foi investigado, resultando em um conjunto de diagramas que demonstram como os componentes se relacionam sob o ponto de vista da comunicação entre eles. A Figura 3.3 é o diagrama principal proposto nesta monografia, que reúne todos os componentes e será descrita nas próximas subseções.

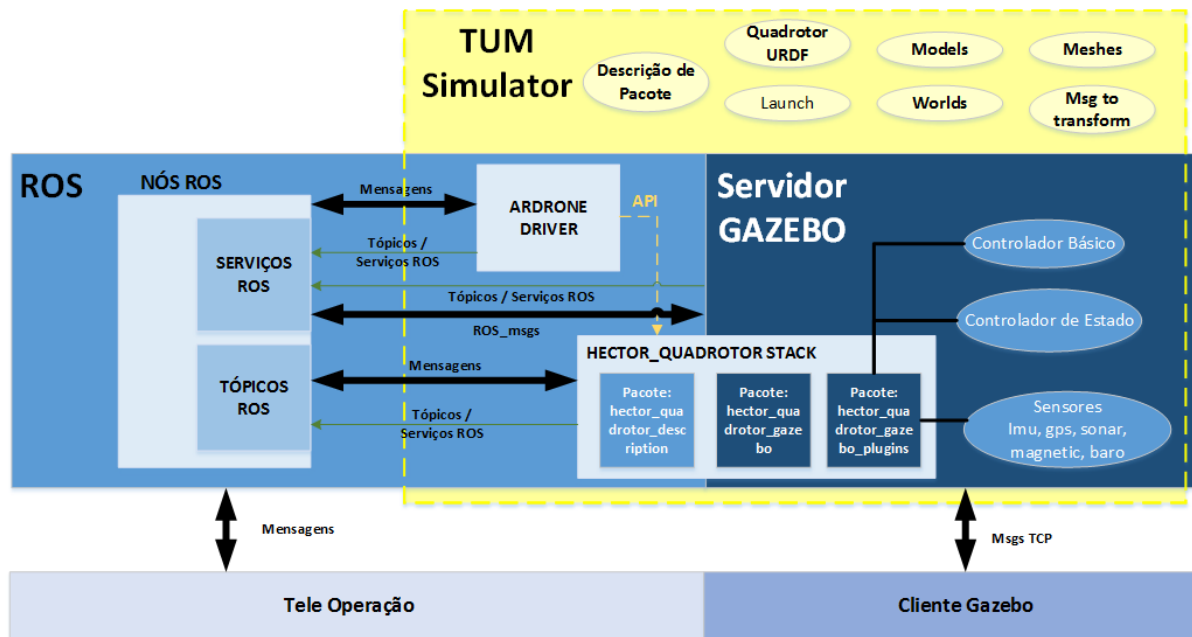


Figura 3.3: Arquitetura básica do TUM Simulator.

A Figura 3.4 mostra de forma macro a relação de dependência entre os componentes. Nela é possível ver que o ROS e o gazebo são os principais componentes, pois todos os demais dependem de um ou de outro. Já o *TUM Simulator* pode ser visto como um grande consumidor dos serviços providos pelos demais componentes, pois tem a função de orquestrar a execução das tarefas necessárias para a simulação dos drones. A pilha Hector Quadrotor também é uma grande consumidora de serviços, pois utiliza os componentes gazebo, ROS e o driver do AR.Drone para fornecer ao TUM Simulator os aspectos físicos do comportamento esperado para um quadrotor. Por fim, o driver do AR.Drone é o que menos tem dependências, sendo utilizado por apenas pelo Hector Quadrotor ao passo que só utiliza os serviços do ROS.

É importante observar que esta é uma arquitetura distribuída, logo conhecer o fluxo de mensagens entre os componentes é importante para entender o seu funcionamento. Além de detalhar a arquitetura mostrada na Figura 3.3, as próximas subseções mostrarão como é este fluxo, qual é o papel de cada componente e como ele é utilizado dentro desta arquitetura.

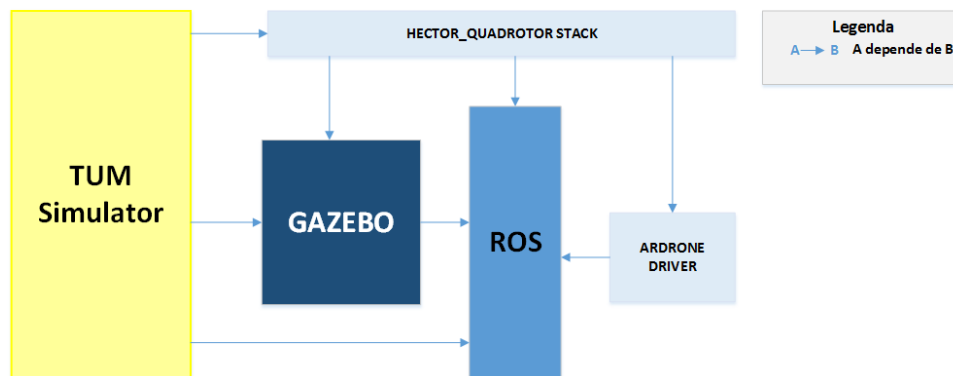


Figura 3.4: Relação de dependência entre os componentes do Simulador AR.Drone.

3.3.1 Papel do ROS na simulação do AR.Drone

O simulador para AR.Drone utiliza o ROS como plataforma robótica para abstração de componentes de hardware e software assim como para o desenvolvimento de aplicações que utilizam o ambiente simulado. Utilizar o ROS para este fim permite que a aplicação desenvolvida possa ser desacoplada do simulador e acoplada no dispositivo físico com pouca ou nenhuma modificação, uma vez que o ROS destina-se originalmente a plataformas robóticas físicas. Como pode ser visto na Figura 3.3, esta arquitetura utiliza a estrutura de comunicação para aplicações distribuídas fornecidas pelo ROS através de tópicos e serviços como ponto principal de comunicação entre os componentes do simulador, abrindo espaço para o desenvolvimento de aplicações distribuídas, sendo esta também uma característica natural do ROS. Os quadrotoros simulados são tele operados utilizando esta mesma estrutura, podendo as mensagens ser geradas diretamente através de APIs disponíveis para diversas linguagens como C++ e Python, ferramentas disponibilizadas pelo próprio ROS, como o comando *rostopic*, ou através de comandos originados em dispositivos externos como controles (joysticks) e dispositivos de automação que são convertidos para mensagens por drivers escritos para o ROS. Outra características do ROS que contribui positivamente para o desenvolvimento do simulador é sua filosofia de colaboração que aproveita a colaboração feita por outros especialistas para o desenvolvimento de pacotes importantes para o desenvolvimento de projetos robóticos. Esta colaboração tem obtido uma grande adesão pela comunidade, sendo isto facilitado pela licença aberta utilizada pelo ROS (BSD license) (Huang e Sturm, 2014; Foundation, 2017).

O componente responsável pela interface entre o quadrotor real e o ROS é o pacote *ardrone_autonomy*, um driver que implementa o SDK 2.0 do AR.Drone no ambiente ROS e dá suporte às versões 1.0 e 2.0 deste equipamento. O objetivo deste driver é permitir o desenvolvimento de aplicações que realizem o controle do drone a partir do ROS. Um ponto de observação importante é que o conjunto ROS + *ardrone_autonomy* é utilizado para o desenvolvimento de muitas pesquisas sobre voo autônomo, reconhecimento de objetos, dentre outros. Sua comunicação com o AR.Drone real é realizada via rede sem fio conforme especificado no SDK. Já o controle para operação pode ser realizado via

tópicos e serviços ROS. Esta estrutura de controle do AR.Drone real pode ser vista na Figura 3.5, criada com base em Huang e Sturm (2014). A API do driver também pode ser utilizada para o desenvolvimento de aplicações, possibilitando a implementação de algoritmos para voos autônomos Monajjemi et al. (2012).

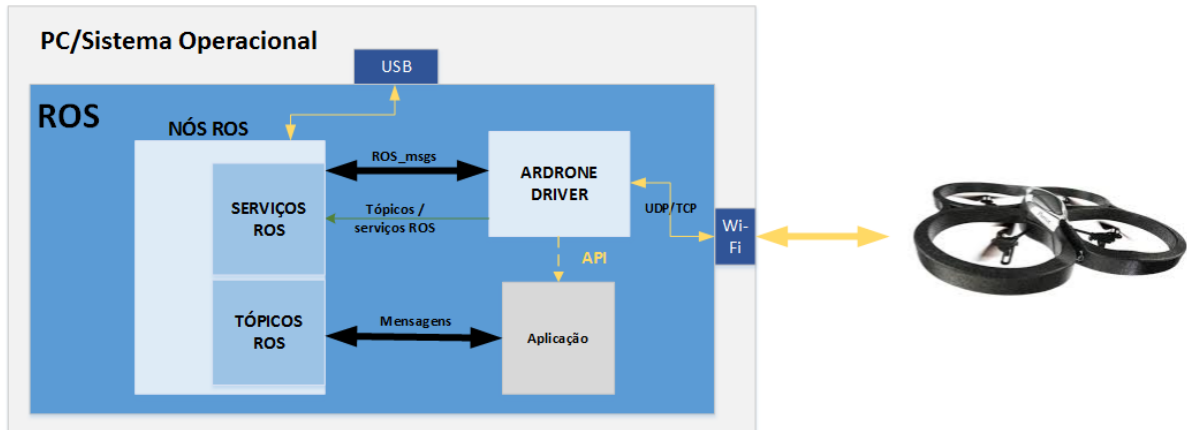


Figura 3.5: Estrutura de controle um AR.Drone real.

Na arquitetura do simulador, o driver é utilizado para a operação do drone virtual via tópicos e serviços, sendo sua API também importada para utilização na pilha de pacotes que é responsável por definir o comportamento do drone, o Hector Quadrotor Stack (Meyer e Kohlbrecher, 2014; Meyer et al., 2012), a ser descrito ainda neste capítulo, substituindo desta forma a conexão TCP/UDP existente no cenário real.

3.3.2 Simulação do AR.Drone no Gazebo

Na arquitetura do simulador, o Gazebo tem a função principal de realizar a simulação de aspectos físicos e dinâmicos do ambiente simulado e dos equipamentos robóticos necessários para a realização dos experimentos. No simulador é utilizado o conjunto de pacotes ROS *gazebo_ros_pkgs* que é responsável pela integração entre o ROS e o Gazebo, sendo responsável por prover as interfaces para a simulação dos robôs no Gazebo. A integração do Gazebo com o ROS funciona baseado em mensagens ROS, serviços e reconfiguração dinâmica (Hsu et al., 2016; OSRF, 2014a). Através dos plug-ins do Hector Quadrotor integrados no Gazebo são simulados o sistema de controle, estados e os diversos sensores sendo seu controle e monitoramento realizado através de mensagens e serviços ROS Meyer (2013). A Figura 3.6 mostra um exemplo desta comunicação utilizando o monitoramento do controle de velocidade para reconfiguração dos parâmetros de velocidade do drone virtual (tópico */cmd_vel*), a publicação das imagens simuladas na câmera frontal (tópico */ardrone/front/image_raw*) e a publicação de dados de navegação (tópico */ardrone/navdata*). Uma vez que esta comunicação é realizada através do ROS, é possível a utilização em sistemas distribuídos através da estrutura já existente no ROS para este fim, sendo este tema abordado em trabalhos futuros.

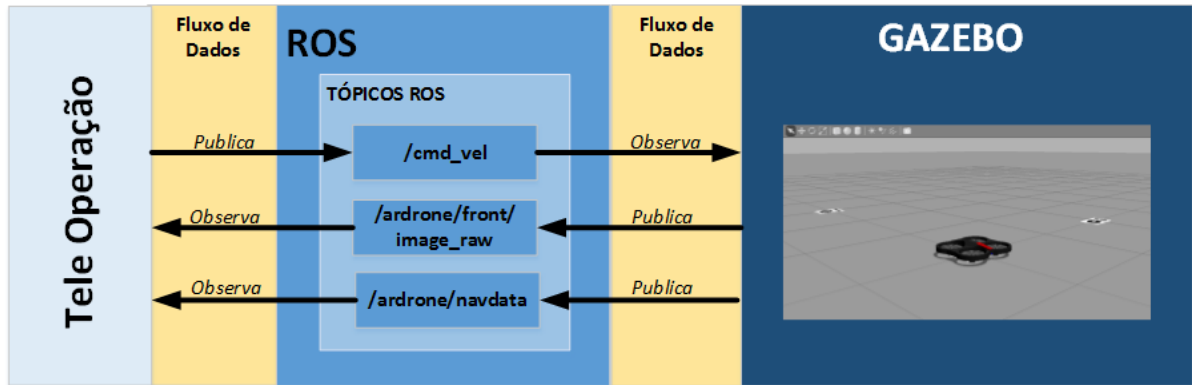


Figura 3.6: Fluxo de dados ROS – GAZEBO.

3.3.3 Hector Quadrotor Stack

Como parte do pacote *tu-darmstadt-ros-pkg*, a pilha Hector Quadrotor é um conjunto de plug-ins baseados no Gazebo e no ROS para simulação de quadrotores, integrando tanto sensores comuns quanto específicos das aeronaves. Seu modelo de funcionamento busca alcançar um alto nível de fidelidade em relação aos quadrotores reais e, para isto, foi realizado um cuidadoso trabalho de construção, teste e comparação dos resultados deste modelo com o funcionamento de quadrotores reais (Meyer et al., 2012). A pilha é composta originalmente de 4 pacotes principais descritos na Tabela 3.4 (Meyer e Kohlbrecher, 2014). No simulador AR.Drone apenas o *hector_quadrotor_teleop*, que controla os objetos simulados via joystick, não é utilizado, sendo todos os demais observados na estrutura de diretórios do simulador. O pacote *hector_quadrotor_description* fornece uma descrição genérica para UAVs. O TUM Simulator estende esta descrição para adequar às definições do AR.Drone. O pacote *hector_quadrotor_gazebo* fornece os arquivos responsáveis pela inicialização de todos os componentes do UAV dentro do ambiente de simulação. Os plugins do pacote *hector_quadrotor_gazebo_plugins* são responsáveis pela simulação do comportamento do drone virtual, gerando as principais tarefas do sistema.

Tabela 3.4: Pacotes componentes da pilha Hector Quadrotor.

Pacote ROS	Descrição
<i>hector_quadrotor_description</i>	Disponibiliza uma descrição URDF genérica para UAVs.
<i>hector_quadrotor_gazebo</i>	Disponibiliza arquivos de inicialização (conhecidos como launch files) e informações de dependência necessários para a simulação do modelo no Gazebo.
<i>hector_quadrotor_teleop</i>	Permite o controle do quadrotor através de um joystick.
<i>hector_quadrotor_gazebo_plugins</i>	Disponibiliza plugins específicos para o funcionamento de UAVs, como sensores específicos, sistemas de controle e de estado.

3.3.4 Considerações Finais

Nesta seção foi descrito simulador utilizado na implementação proposta nesta monografia para simular o AR.Drone, descrevendo seus componentes, como estes funcionam, como eles se interrelacionam e como funciona sua comunicação. Também foi descritos o componente responsável por simular o comportamento do equipamento físico, sendo indicados os arquivos que implementam este comportamento e os tópicos ROS relacionados a eles. Os diagramas e tabelas apresentados aqui contribuem para o melhor entendimento do funcionamento deste simulador e quais componentes podem ser modificados para alterar o seu comportamento. Um exemplo prático desta aplicação é o teste de novos modelos dinâmicos antes destes serem implementados no equipamento físico.

ABORDAGEM PROPOSTA

4.1 INTRODUÇÃO

O presente trabalho tem como principal objetivo construir uma arquitetura em nuvem para a simulação de ambientes para pesquisas de voos autônomos com quadrotoros contribuindo para o Projeto DaaS (Drones as a Service), atualmente em desenvolvimento no LaSiD. Para atingir este objetivo foi realizada uma pesquisa em busca de trabalhos já realizados e a partir desta pesquisa foi desenhada uma arquitetura que levasse em consideração as principais dificuldades dos pesquisadores do LaSiD em relação à realização de experimentos com drones reais:

- a) Os pesquisadores envolvidos nos projetos do laboratório utilizam drones reais para seus experimentos, sendo que cada equipamento é utilizado exclusivamente por um pesquisador, o que acarreta limites de pesquisadores ou possíveis conflitos de horário caso haja compartilhamento de equipamento. A duração dos experimentos realizados está limitada ao uso das baterias que duram no máximo 18 minutos;
- b) Para melhorar este cenário foram adquiridas baterias adicionais, mas não são suficientes para um período mais prolongado de experimentos (duas horas, por exemplo);
- c) Os experimentos são realizados nas dependências do laboratório, logo a área física é limitada sendo necessário utilizar o ambiente externo para a realização de testes que necessitem mais espaço, o que insere mais algumas dificuldades como falta de rede cabeada para acesso à rede da instituição caso necessário, maior poluição do sinal de rede sem fio e maior risco de acidentes com o equipamento;
- d) O pesquisador precisa deslocar-se até o laboratório sempre que for utilizar o drone, restringindo o horário/tempo de pesquisa, bem como o apoio de outros pesquisadores que estejam em outras localidades.

As próximas seções deste capítulo apresentarão uma proposta de arquitetura em nuvem que oferta ambientes de simulação robótica como serviço. Também é proposto um modelo para a implementação desta arquitetura e, em seguida, é apresentada uma implementação baseada em Software Livre realizada no LaSiD para atender às necessidades das pesquisas com drones realizadas pelos pesquisadores deste laboratório.

4.2 TRABALHOS CORRELATOS

A plataforma REALabs foi proposta por [Cardozo et al. \(2010\)](#) com o objetivo de favorecer a integração entre robôs móveis e dispositivos conectados a rede através de uma arquitetura de computação distribuída acessível via Web. [Agostinho et al. \(2011\)](#) estende este trabalho propondo um ambiente de computação em nuvem para aplicações robóticas em rede onde a plataforma REALabs é oferecida como serviço para o desenvolvedor. Em ([Cardozo et al., 2010](#)), os desenvolvedores de aplicações robóticas executam seus códigos localmente e controlam os robôs disponíveis no laboratório através da plataforma REALabs via Internet. Apesar deste modelo funcionar bem para uma gama de aplicações, não é interessante para as que são sensíveis a atrasos de comunicação e necessitam de alto poder de processamento. [Agostinho et al. \(2011\)](#) enfrenta este desafio transferindo a execução da aplicação para máquinas virtuais que fazem parte da mesma rede onde se localizam os robôs e executam em servidores com um maior poder computacional. Para a presente pesquisa é necessária uma arquitetura que leve em consideração o controle de um robô virtual, no caso drones, em um ambiente simulado, o que não é abordado em nenhum destes trabalhos. Assim, o trabalho de [Agostinho et al. \(2011\)](#) foi utilizado como base para o desenho da arquitetura aqui proposta e modificado para fornecer simulações robóticas.

[Barham et al. \(2003\)](#) apresenta o Xen, um gerenciador de máquinas virtuais (VMM) de código aberto para a plataforma x86 que permite a execução simultânea de máquinas virtuais com Sistemas Operacionais hóspedes diferentes, garantindo o isolamento de recursos. Para alcançar uma baixa sobrecarga na utilização dos recursos computacionais, são utilizadas máquinas virtuais para-virtualizadas¹. [Xi et al. \(2014\)](#) estende o Xen ao propor o RT-Xen, um escalonador para a plataforma de virtualização Xen que tem como objetivo atender tanto a máquinas virtuais que executem tarefas com restrições de tempo real quanto de propósito geral. Para permitir a simulação de modelos de tarefas presentes no modelo de quadrotor simulado utilizado neste trabalho foi adotado este escalonador na implementação da arquitetura em nuvem proposta.

[Parrot Inc \(2010\)](#) lançou o AR.Drone, um quadrotor de baixo custo baseado na plataforma ARM e no sistema operacional de código aberto Linux com foco no mercado de jogos com realidade aumentada. Apesar do foco na comunidade de jogos, este equipamento é muito utilizado em pesquisas científicas por ter uma boa relação custo/benefício e permitir o embarque de aplicações e extensão de hardware. [Bristeau et al. \(2011\)](#) apresenta a tecnologia de controle e navegação embarcada neste equipamento que o permite manter uma boa estabilidade utilizando sensores e atuadores de baixo custo. O AR.Drone é o equipamento utilizado nos laboratórios do LaSiD e é o objeto a ser simulado na Nuvem

¹Ver Subseção 2.2.1.

de Simulação proposta neste trabalho. Também é apresentado na [Capítulo 3](#) um estudo acerca do funcionamento do AR.Drone e seu modelo de tarefas que levou em consideração o trabalho de Bristeau, além dos trabalhos de [Bergner e Regelungstechnik \(2012\)](#), [Chaperon e Pierre \(2011\)](#), [PaparazziUAV \(2016\)](#), [Federal Communications Commission \(2010\)](#), [Perquin \(2012\)](#), [Piskorski et al. \(2012\)](#), [Krajník et al. \(2011\)](#).

[Quigley et al. \(2009\)](#) apresenta o ROS, um framework de código aberto projetado para o desenvolvimento de aplicações robóticas que se diferencia dos demais lançados até aquele momento por permitir a contribuição da comunidade no desenvolvimento e compartilhamento de componentes de software para a plataforma. Sua arquitetura utiliza a topologia ponto-a-ponto (peer-to-peer) para a comunicação entre nós e serviços, pois lida melhor com redes compostas por diversos meios (ex. rede com e sem fio). Na arquitetura do ROS é garantido o suporte a múltiplas linguagens de programação, uma vez que sua especificação é feita no nível do gerenciamento das mensagens que são trocadas entre nós e serviços, implementa um microkernel que provê as ferramentas para fornecer as funcionalidades necessárias para o desenvolvimento das aplicações e motiva a produção de códigos baseados em bibliotecas independentes, facilitando assim o re-uso e gerando também binários menores. O ROS é utilizado neste trabalho como plataforma integrante do Ambiente de Simulação disponibilizado para que os desenvolvedores possam executar suas aplicações. [Monajjemi et al. \(2012\)](#) amplia o ROS ao implementar neste framework um driver para o quadrotor AR.Drone, da Parrot, baseado no SDK oficial deste equipamento.

[Koenig e Howard \(2004\)](#) desenvolveram o Gazebo, um simulador multi-robôs de código aberto que supriu as necessidades latentes à época de realizar a simulação de robôs em ambientes externos e simular respostas realísticas de sensores. Os objetos simulados possuem aspectos físicos como massa, velocidade, fricção dentre outros atributos. [Hsu et al. \(2016\)](#) estendeu este trabalho desenvolvendo o plugin *gazebo_ros_pkgs* utilizado neste trabalho que realiza a integração da plataforma Gazebo com o framework ROS. [Meyer e Kohlbrecher \(2014\)](#) utilizou o conjunto ROS + Gazebo para implementar um conjunto de plugins para simulação de quadrotores denominado Hector Quadrotor Stack. Este, por sua vez, é utilizado por [Huang e Sturm \(2014\)](#) para criar o TUM Simulator, o simulador do quadrotor AR.Drone que é oferecido como serviço na plataforma de Nuvem de Simulação que é proposta no presente trabalho.

4.3 NUVEM DE SIMULAÇÃO E MODELO DE IMPLEMENTAÇÃO

A arquitetura proposta neste trabalho é composta por (ver [Figura 4.1](#)): Ator *Usuário Comum*, que é o usuário final da nuvem atuando como utilizador dos ambientes aos quais tem permissão para acesso. O ator *Usuário Administrador* é o responsável pela administração e manutenção de todos os ambientes, tendo o mesmo papel que um usuário comum somado ao papel administrativo. A *Interface do Usuário* é a camada que interage com os usuários e administradores realizando a tradução das solicitações destes atores para chamadas à *API de Gerenciamento de Ambientes*. A *API de Gerenciamento do Ambiente* é a camada que disponibiliza as operações de gerenciamento da *Camada de Gerenciamento* para a *interface do usuário*. Nem todas as operações existentes nas camadas abaixo da

API são disponibilizadas para o usuário. A *Camada de Validação de Segurança* realiza as verificações de acesso às operações solicitadas pelo usuário, permitindo que as operações só sejam executadas nos ambientes que o solicitante tem permissão. A *Camada de Gerenciamento* define todas as operações que auxiliam o gerenciamento do ambiente. Os modelos utilizados no momento da criação de um ambiente podem ser dos tipos *modelo de máquina virtual* e *modelo de cenário robótico*, sendo gerenciados pela camada de *Modelos*. A *Camada de Virtualização* é responsável pelos ambientes virtualizados que contêm o ambiente utilizado para a simulação e execução de outras tarefas importantes para os experimentos. O *Filtro de Segurança* é a camada que filtra os pacotes de rede, separando os que possuem permissão para trafegar para os dispositivos de rede virtualizados das máquinas virtuais dos que não possuem. A *Camada de Simulação* é onde as aplicações robóticas serão executadas e utilizarão as ferramentas de simulação que estão instaladas nesta camada. Cada um dos componentes da arquitetura proposta serão apresentados nas subseções a seguir.

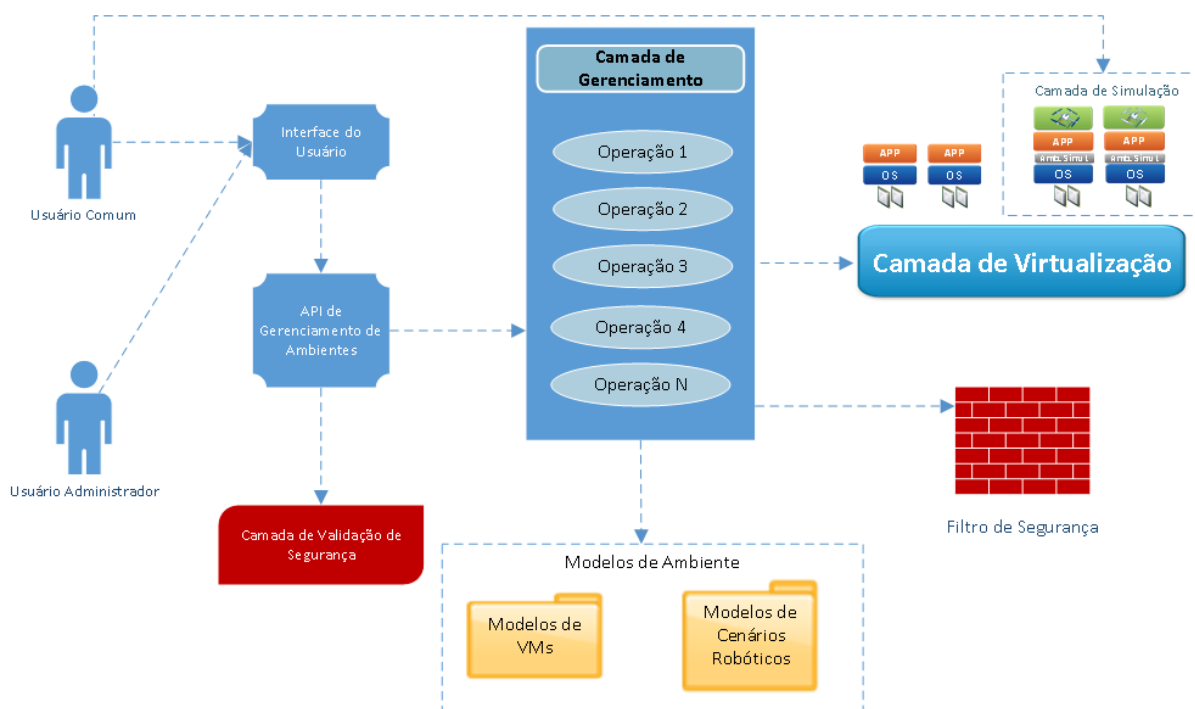


Figura 4.1: Arquitetura da Nuvem de Simulação.

Fonte: Autor.

4.3.1 Interface do Usuário

A Interface do Usuário é o componente responsável pela interação da Nuvem com o usuário. Assim como no modelo de [Agostinho et al. \(2011\)](#), o controle sobre a Nuvem é garantido, mantendo o usuário sem acesso direto às operações que podem ser executadas pelas camadas inferiores. Esta camada traduz as requisições dos usuários para a interface disponibilizada pela API de Gerenciamento de Ambientes, descrita na [Subseção 4.3.2](#).

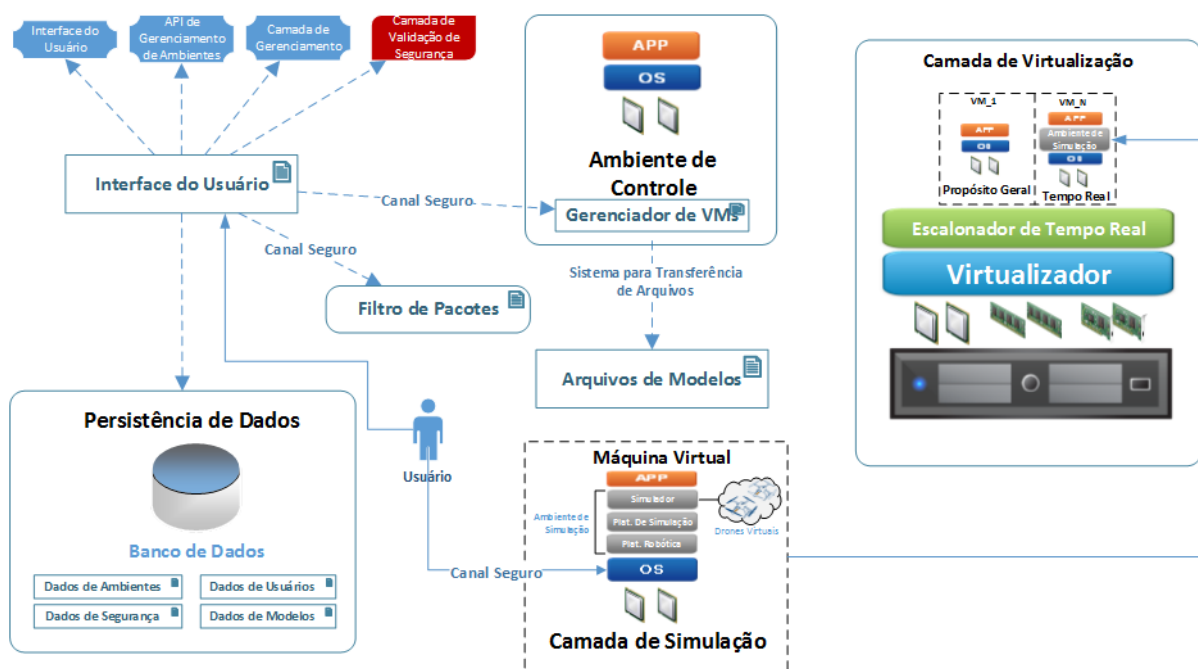


Figura 4.2: Modelo proposto para implementação da Nuvem de Simulação.

Fonte: Autor.

As seguintes operações estão disponíveis na implementação realizada por este trabalho e serão descritas na Subseção 4.4.3:

Listar Ambientes. Mostra ao usuário uma lista dos ambientes² disponíveis.

Visualizar Ambiente. Mostra informações sobre o ambiente solicitado pelo usuário.

Listar Modelos. Mostra ao usuário uma lista dos modelos disponíveis.

Visualizar Template. Mostra informações sobre o modelo solicitado pelo usuário.

Iniciar um Ambiente. Inicia a máquina virtual responsável pelo ambiente ambiente solicitado pelo usuário.

Parar um Ambiente. Para a máquina virtual responsável pelo ambiente solicitado pelo usuário.

Criar Ambiente. Realiza a criação de um ambiente³.

Remover Ambiente. Realiza a exclusão de um ambiente existente³.

Aplicar Cenário a Ambiente. Aplica um modelo de cenário disponível no sistema ao ambiente especificado pelo usuário³.

²Na *Interface do Usuário* um ambiente é uma instância da camada de simulação, conforme Figura 4.2. Esta camada é descrita na Subseção 4.4.7

³Esta operação é realizada por um usuário com o perfil de *Usuário Administrador*.

A implementação da *Interface do Usuário* terá como dependência a API, pois a forma de chamada das operações disponibilizadas pela API é que vai indicar os aspectos de implementação da interface do usuário. Em relação ao usuário, existem dois tipos que devem ter interação com esta camada: a) um usuário comum, que realiza operações triviais sobre os ambientes aos quais tem acesso; b) um usuário administrador, que realiza operações de administração dos ambientes além das de um usuário comum. O diagrama de sequência da [Figura 4.3](#) mostra como se dá o tratamento de uma solicitação do usuário e a interação da Interface do Usuário com a API de Gerenciamento de Ambientes. O processo inicia com a solicitação realizada pelo usuário através da Interface do Usuário, fornecendo além da operação a ser realizada os parâmetros necessários para sua execução. Os dados fornecidos são então validados e, tendo o usuário solicitado uma operação existente na interface e fornecido todos os dados obrigatórios, a execução da operação é solicitada à API de Gerenciamento de Ambientes. A API também faz a validação e, caso os dados informados sejam corretos, solicita que a Camada de Validação de Segurança informe se este usuário está autorizado a executar a operação no ambiente fornecido. Caso o usuário esteja autorizado, a operação é executada pela Camada de Gerenciamento e o resultado da operação é retornado para a Interface do Usuário e, posteriormente, ao usuário. Caso o usuário não esteja autorizado, o resultado da operação é definido como "Não permitido" e é retornado seguindo o mesmo fluxo do caso anterior. No modelo de implementação proposto a Interface do Usuário, a API de Gerenciamento de Ambientes e as camadas de Gerenciamento e Validação de Segurança são implementadas em um mesmo artefato (ex.: no mesmo executável binário), facilitando o desenvolvimento e implantação (ver [Figura 4.2](#)).

4.3.2 API de Gerenciamento de Ambientes

A API de gerenciamento de ambientes tem como objetivo desacoplar a Camada de Gerenciamento da Interface do usuário e da Camada de Validação, simplificando a implementação destas camadas. Esta API garante ao modelo de ([Agostinho et al., 2011](#)) a flexibilidade necessária ao desenvolvimento de Interfaces de Usuário customizadas ou também plugins de integração para outras plataformas de gerenciamento de Nuvem já consolidadas no mercado como o OpenStack, OpenNebula e Eucalyptus. Antes de solicitar à Camada de Gerenciamento que execute a operação requisitada pelo usuário duas validações importantes são realizadas por esta camada: a) validar as solicitações da Interface do Usuário para garantir que todos os dados necessários para continuar o fluxo da execução sejam fornecidos corretamente; b) verificar se o usuário tem permissão para executar esta operação realizando uma chamada à Camada de Validação de Segurança, descrita na próxima seção. O diagrama de sequência da [Figura 4.3](#) demonstra a relação entre a API e as demais camadas. Esta camada relaciona-se com o componente de Persistência de Dados do modelo de implementação proposto neste trabalho que permite a gravação, recuperação e eliminação dos dados utilizados na Nuvem em um banco de dados adequado para a implementação realizada. Faz parte do conjunto de dados administrados por este componente os dados relacionados com os ambientes, usuários, modelos e segurança. A API utiliza os dados de ambiente e de modelos durante suas operações.

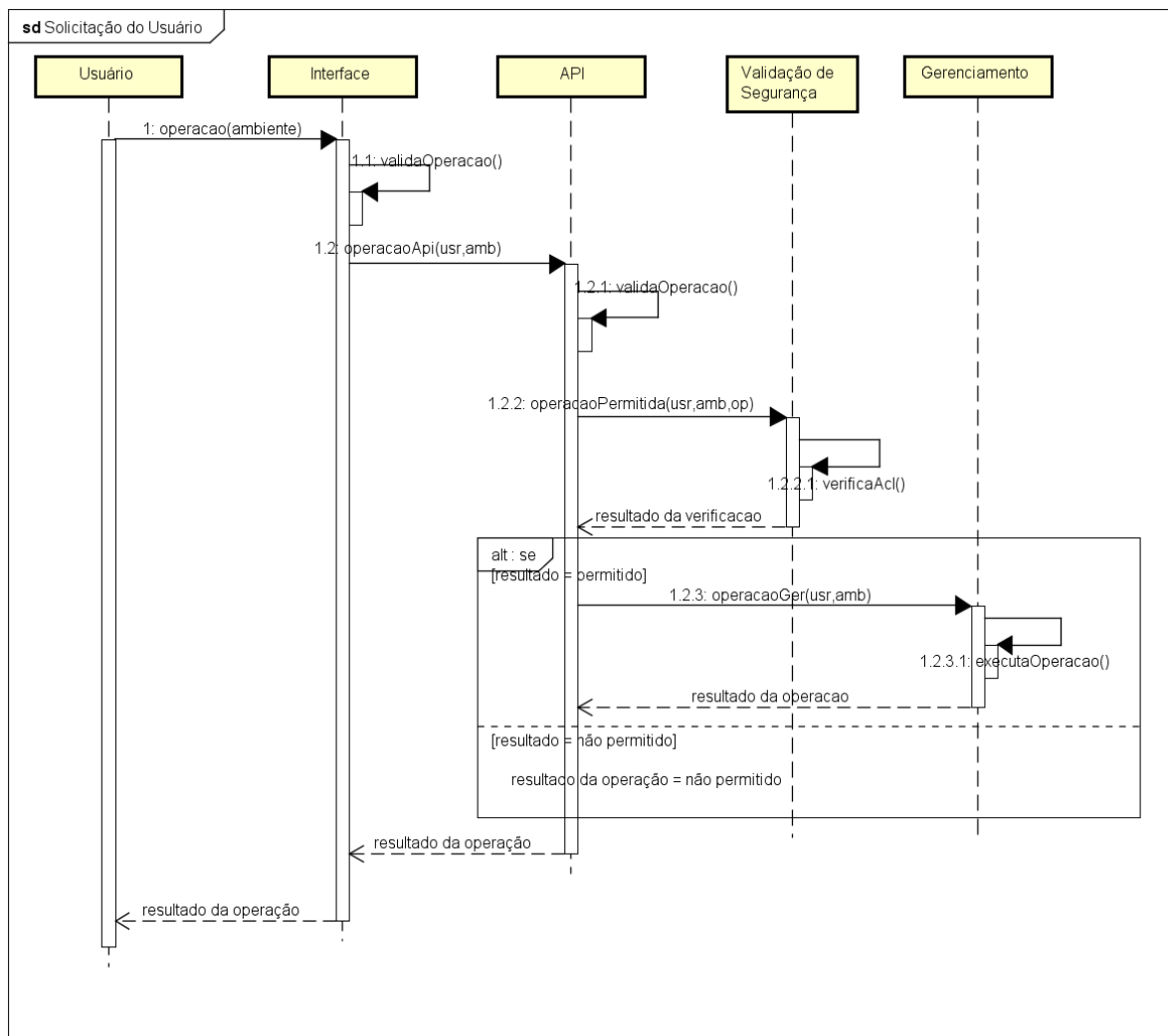


Figura 4.3: Diagrama de sequência: Solicitação do usuário.

Fonte: Autor.

4.3.3 Camada de Validação de Segurança

A Camada de Validação de Segurança é responsável por verificar se uma operação solicitada pelo usuário pode ser executada no ambiente solicitado. As verificações básicas que esta camada precisa realizar são:

- Verificar se o usuário está autenticado;
- Verificar se o usuário existe no sistema;
- Verificar se a operação solicitada existe no sistema;
- Verificar se o usuário pode executar aquela operação no ambiente solicitado;
- Registrar as solicitações em logs para auditoria.

Outras verificações referentes a esta relação usuário-ambiente-operação podem ser adicionadas à camada para atender aos requisitos específicos de segurança. No modelo de implementação proposto esta camada utiliza a Camada de Persistência de Dados para manipulação dos dados dos ambientes, usuários e segurança.

4.3.4 Camada de Gerenciamento

A Camada de Gerenciamento é responsável por realizar todas as operações que fazem parte do gerenciamento da nuvem, orquestrando os seus componentes para gerir o ciclo de vida dos ambientes de simulação. Operações como criação, modificação e remoção dos ambientes, inicialização e parada de máquinas virtuais, gerenciamento dos modelos de VMs e cenários, dentre outras, são realizadas por esta camada. Suas principais interações ocorrem com os seguintes componentes:

- a) API de Gerenciamento de Ambientes: Fornece uma fachada por onde as operações solicitadas pelo usuário chegam para serem executadas.
- b) Camada de Virtualização: Realiza solicitações para gerenciamento das máquinas virtuais vinculadas aos ambientes de simulação.
- c) Camada de Persistência de Dados: Seleciona e manipula dados necessário para a gestão do ciclo de vida dos ambientes de simulação.
- d) Gerenciador de VMs: Solicita ao gerenciador a realização de operações nas máquinas virtuais dos ambientes gerenciados.
- e) Filtro de pacotes: Iniciar ou para o filtro de segurança para cada ambiente no momento em que este é iniciado ou parado, respectivamente.

No modelo de implementação proposto o conjunto de ferramentas utilizado por esta camada para realizar o gerenciamento das VMs existentes na Camada de Virtualização é denominado Gerenciador de VMs e está localizado no Ambiente de Controle, que é composto de uma Máquina Virtual de propósito geral com um Sistema Operacional compatível com o Gerenciador de VMs. O Ambiente de Controle é um artefato que pode apoiar não apenas a Camada de Gerenciamento, mas também pode servir a outras camadas. Para garantir a segurança no envio de dados sensíveis (como dados de autenticação, por exemplo) a comunicação com este componente é realizada após o estabelecimento de um canal seguro.

4.3.5 Modelos de Ambientes (Templates)

Um Modelo de Ambiente é composto de três componentes: um Modelo de VMs, que define as configurações de hardware e software de uma máquina virtual, um ou mais Modelos de Cenários Robóticos, que definem os cenários de simulação que estarão configurados na Camada de Simulação, e dos metadados do Modelo de Ambiente, cujo objetivo principal é referenciar os modelos envolvidos na sua composição. Os Modelos de Ambientes

disponíveis são gerenciados pela Camada de Gerenciamento, sendo utilizados na criação de um novo ambiente. Enquanto um Modelo de VM só é utilizado na criação de um ambiente, um Modelo de Cenário Robótico pode ser associado a um ambiente em qualquer momento posterior à sua criação, pois este é um artefato que tem o objetivo de permitir a adequação dos ambientes a novas necessidades de simulação dos projetos aos quais eles servem.

No modelo de implementação da **Figura 4.2** os Modelos de VMs e os Modelos de Cenários Robóticos são definidos por arquivos mantidos no Repositório de Modelos e disponibilizados através de um sistema que implementa protocolos apropriados para a transferência de arquivos (ex.: NFS, CIFS ou FTP). Estes arquivos são inseridos nas máquinas virtuais após sua criação, podendo também ser executados para a instalação do novo cenário. Os dados sobre os modelos são gravados e manipulados na Camada de Persistência de Dados.

4.3.6 Filtro de Segurança

Este componente implementa configurações de segurança que podem filtrar tráfego de rede, utilização de serviços ou outros aspectos de acordo com os recursos fornecidos pela camada de virtualização⁴ ou pelo sistema operacional instalado na máquina virtual. Os filtros são aplicados no nível de Ambiente de Simulação, ou seja, cada ambiente tem configurações específicas implementadas pelo filtro de segurança que são ativados na inicialização do ambiente e desativados no seu desligamento. No modelo de implementação da **Figura 4.2**, o Filtro de Segurança é implementado através de um filtro de pacotes de rede. Este filtro deve estar localizado em um ponto da topologia de rede que permita o isolamento da Camada de Virtualização. A gerência deste filtro é realizada pela Camada de Gerenciamento através de um canal de comunicação seguro, sendo ativado e desativado na inicialização e parada, respectivamente, de cada ambiente.

4.3.7 Camadas de Virtualização e Simulação

A camada de virtualização é o destino de todas as solicitações realizadas pelo usuário, pois ela irá gerenciar a Camada de Simulação. A Camada de Simulação é responsável por abrigar o Ambiente de Simulação, uma plataforma que o usuário utilizará para realizar seus experimentos permitindo a obtenção de resultados simulados da sua pesquisa antes da realização dos testes em drones reais. Esta camada é formada por máquinas virtuais que contem um Sistema Operacional hospede e um Ambiente de Simulação. O Ambiente de Simulação contém as plataformas e ferramentas necessárias para que os pesquisadores possam executar as aplicações robóticas. O planejamento da implementação dos três componentes citados deve levar em consideração os seguintes aspectos:

- a) Requisitos de tempo-real: Conforme pode ser observado na **Capítulo 3**, existem tarefas em um drone real que precisam cumprir requisitos de tempo-real logo, para

⁴Como exemplo desses recursos de segurança podemos citar o Xen Security Modules (XSM), uma camada de segurança que permite um controle mais granular das VMs e suas interações com o hypervisor, dispositivos, outras VMs e etc.

a plataforma proposta neste estudo, a correta execução destas tarefas é importante. O sistema operacional hospede das máquinas virtuais de tempo real que suportam o Ambiente de Simulação (ver Figura 4.2) também deve ser adequado à execução destas cargas.

- b) Ambiente de simulação adequado a aplicações robóticas: O ambiente disponibilizado para o pesquisador precisa ter uma boa aderência à execução das aplicações robóticas previstas nos experimentos que utilizarão a plataforma.
- c) Similaridade com o ambiente real: Para o desenvolvimento de pesquisas reais, o ambiente precisa disponibilizar uma simulação de equipamentos e cenários que tenham similaridade com os equipamentos e cenários reais.

O modelo de implementação proposto neste trabalho prevê um sistema de virtualização que suporte a utilização de pelo menos um escalonador capaz de atender a demandas de Tempo Real e Propósito Geral conforme discutido na Subseção 2.2.2. O virtualizador deve ser de tipo 1⁵ para minimizar a sobrecarga imposta pela camada de virtualização (VMM) e permitir configurações individuais e flexíveis de recursos para cada máquina virtual utilizada para que os cenários possam atender aos requisitos das pesquisas do usuário⁶. O Sistema Operacional instalado nas máquinas virtuais que fazem parte da Camada de Simulação deve ser preferencialmente para-virtualizado também para minimizar a sobrecarga no VMM.

O Ambiente de Simulação previsto neste modelo de implementação é composto de três componentes:

- a) Plataforma Robótica: fornece bibliotecas e ferramentas que darão suporte à execução das aplicações robóticas.
- b) Plataforma de Simulação: fornece suporte para a simulação dos aspectos visuais e físicos de cenários e dispositivos robóticos.
- c) Simulador: fornece os cenários, drones e outros dispositivos robóticos implementados na plataforma de simulação.

Estes três componentes estão disponíveis para utilização das aplicações executadas pelo usuário, cujo acesso a este ambiente é realizado através de um canal de comunicação seguro. Os Modelos de Cenários Robóticos atuam nestes componentes para que novos cenários, drones e outros dispositivos robóticos possam ser adicionados ao Ambiente de Simulação permitindo que os ambientes consigam se adaptar a mudanças que ocorram durante o desenvolvimento das pesquisas.

⁵Instalado diretamente no hardware como seu sistema operacional - ver Seção 2.2)

⁶Como exemplo, Lee et al. (2010), Xi et al. (2011) utilizaram em suas pesquisas configurações de afinidade de CPU e desativação de dispositivos para garantir que os experimentos fossem executados em um ambiente adequado.

Todos os componentes da arquitetura da Nuvem de Simulação apresentada na Figura 4.1 foram apresentados e discutidos nesta seção, bem como a proposta para o modelo de implementação desta Nuvem apresentada na Figura 4.2. Os pontos discutidos até aqui dão subsídios para a implementação da Nuvem de Simulação realizada no presente trabalho e descrito na próxima seção.

4.4 IMPLEMENTAÇÃO

A nuvem de simulação proposta neste trabalho foi implementada no Laboratório de Sistemas Distribuídos da Universidade Federal da Bahia com o objetivo de atender às demandas de pesquisas na área de navegação autônoma de quadrotores e minimizar algumas limitações que a necessidade de utilização apenas dos drones reais traz para os pesquisadores.

A implementação da nuvem foi realizada conforme o diagrama da Figura 4.4 em um servidor Dell R610 com dois processadores Xeon E5506 2.13 GHz de 4 núcleos, 16 GB de memória RAM, dois discos rígidos de 146 GB espelhados por hardware e quatro dispositivos de rede com vazão máxima de 1 GBit/s.

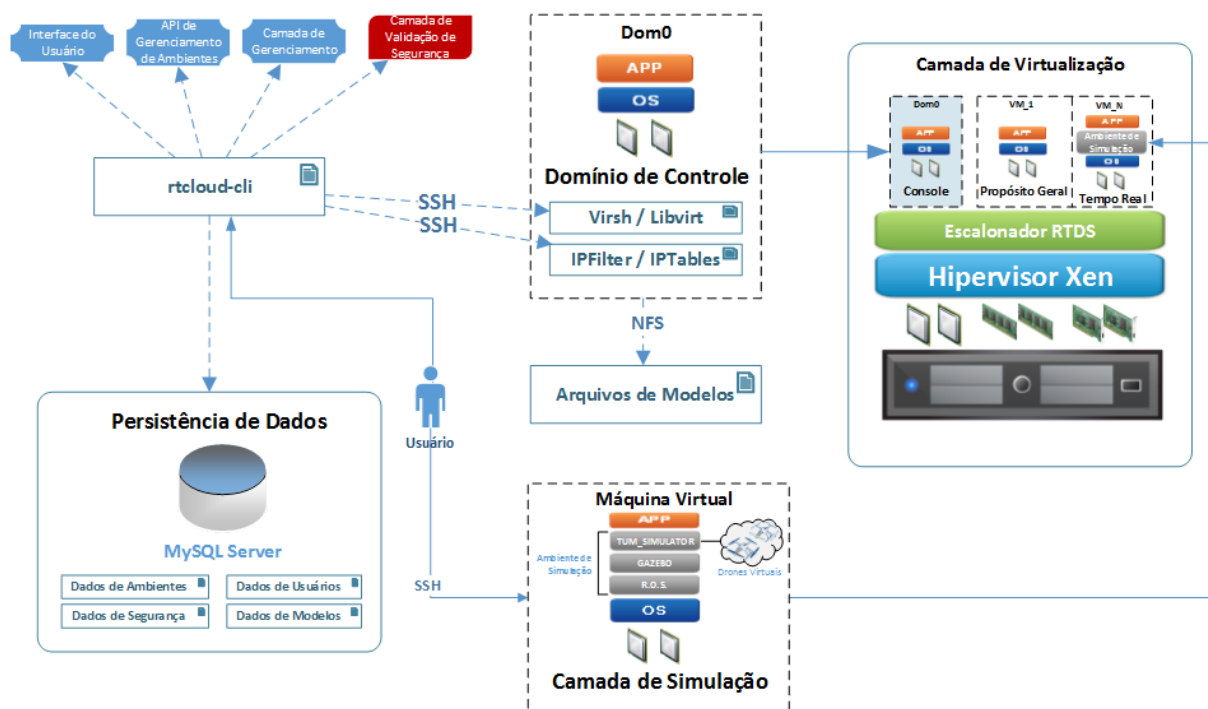


Figura 4.4: Diagrama de implementação da nuvem de simulação do LaSiD.

Fonte: Autor.

A Camada de Virtualização, descrita na Subseção 4.4.6, foi baseada no virtualizador Xen (Xen Project Community, 2016) versão 4.6 com escalonamento de vCPUs realizado pelo escalonador RT-Xen (Xi et al., 2014)⁷. O Gerenciador de VMs ou *domínios*, como é

⁷O projeto Xen incorporou o RT-Xen à sua lista de escalonadores sob o nome de RTDS Scheduler, disponível a partir da versão 4.5 do virtualizador.

descrito na literatura do Xen, é realizada pelo *libvirt*⁸, uma API para administração de servidores de virtualização que permite a execução de operações para criação, remoção, modificação, inicialização e parada de VMs, dentre outras possíveis. Como filtro de segurança foi utilizado o IPFilter/IPTables⁹, disponível para sistemas operacionais Linux a partir da versão 2.4. Estas duas ferramentas estão instaladas no Ambiente de Controle que, por sua vez, faz parte do Domínio de Controle (Dom0) do Xen e acessa os arquivos de modelos via protocolo Network File System (NFS). A camada de simulação disponibiliza o ROS (Foundation, 2017) como base para o desenvolvimento de aplicações robóticas e a plataforma de simulação de ambientes robóticos Gazebo (OSRF, 2014a). Uma aplicação foi desenvolvida utilizando a linguagem Python para o gerenciamento da Nuvem. Denominada *rtcloud-cli*, inclui como componentes a Interface do Usuário, a API de Gerenciamento de Ambientes, a Camada de Gerenciamento e a Camada de Validação de Segurança. Esta aplicação utiliza o banco de dados MySQL (Oracle Corporation., 2017) para persistência dos dados relativos aos ambientes e seus usuários, dados da Camada de Validação de Segurança e metadados dos modelos de ambientes e cenários robóticos. A aplicação realiza as operações na Camada de Virtualização utilizando o protocolo de acesso remoto seguro Secure Shell (SSH). Nas próximas subseções esta implementação será descrita com mais detalhes.

4.4.1 *rtcloud-cli* - Gerenciamento de Ambientes

A aplicação *rtcloud-cli* foi implementada conforme a Interface do Usuário descrita na Subseção 4.3.1, portanto, incluindo as camadas de Gerenciamento, Validação de Segurança, Interface de Usuário e Validação de Segurança. Como pode ser visto no modelo de implementação (ver Figura 4.2), estas camadas têm como dependência o acesso à Camada de Persistência, ao Filtro de Pacotes e ao Gerenciador de VMs. A camada de persistência é acessada via protocolo TCP na porta 3306, padrão do MySQL, e o acesso ao Filtro de Pacotes e Gerenciador de VMs é realizado acessando o Domínio de Controle via protocolo seguro SSH na porta padrão (TCP 22). Seu desenvolvimento foi realizado utilizando a linguagem de programação Python versão 3.5 e levou em consideração alguns aspectos de engenharia de software para o levantamento de requisitos, planejamento e implementação com objetivo de alcançar uma maior qualidade no software desenvolvido. A seguir estes aspectos serão descritos.

4.4.1.1 Elicitação de Requisitos A Figura 4.5 apresenta o diagrama de casos de uso do sistema, seguindo os modelos propostos nas Seções 4.1 e 4.3. Os atores deste diagrama são o Sistema, o Usuário e o Administrador. O *Sistema* é o *rtcloud-cli* e executa casos de uso necessários para o funcionamento do sistema, como carregar o arquivo com Parâmetros de Configuração e realizar a verificação de permissões do usuário. O *Usuário* é alguém que está utilizando o sistema para operacionalizar a Camada de Simulação através da execução de tarefas como criar, remover, visualizar, iniciar e parar ambientes, exibir listas de modelos e ambientes e aplicar cenários a ambientes. E o Administrador

⁸Disponível em <http://libvirt.org/index.html>

⁹A descrição deste filtro de pacotes está disponível em <https://www.netfilter.org>

é uma especialização do ator *Usuário* que realiza operações para administrar o sistema, como cadastrar, remover, listar e visualizar modelos, cadastrar, remover, listar, visualizar perfis de firewall, realizar a associação dos perfis aos ambientes, cadastrar, remover, listar e visualizar *Access Control Lists* (ACLs)¹⁰. Dentre os casos de uso levantados, o escopo deste trabalho considera a implementação dos casos de uso ligados ao usuário e ao sistema, conforme pode ser visto na **Figura 4.6**.

A partir do Diagrama de Casos de Uso, foram selecionados os casos de maior complexidade e que tinham mais interações entre os componentes da nuvem para a criação dos seus diagramas de sequência, disponíveis no Apêndice A.1. Estes diagramas, a descrição da arquitetura e o modelo de implementação da Nuvem de Simulação (ver **Seção 4.3**) foram analisados para concluir a elicitação de requisitos funcionais e não-funcionais do *rtcloud-cli*. Os requisitos funcionais foram agrupados nos níveis de Usuário e Sistema, descritos a seguir.

Nível de Usuário. No nível de Usuário é possível a realização de operações de criação, remoção, inicialização e parada de um Ambiente Robótico específico pelo ator *Usuário*. Um ambiente robótico é composto de no mínimo uma VM criada a partir de um Modelo de VM disponível na Nuvem, um drone virtual e um Modelo de Cenário Robótico associado ao ambiente. Um *Usuário* também pode listar ou visualizar os ambientes e Modelos de Cenário disponíveis para sua utilização. Ainda a nível de usuário, o ator *Administrador*, por ser uma especialização do ator Usuário, herda todos os requisitos pertinentes a este ator, podendo ainda: a) criar e remover Modelos de VMs e Cenários Robóticos, ACLs e perfis de firewall; b) listar todos os perfis de firewall e ACLs existentes no ambiente e c) associar e desassociar perfis de firewall aos ambientes de simulação.

Nível de Sistema. No nível do Sistema um Ambiente Robótico tem uma identificação única e um nome único além da composição já descrita para o nível de Usuário. Durante a criação deste ambiente deverá ser associado a este um Cenário Robótico. Um cenário robótico é composto de arquivos e configurações que definem as condições de simulação do ambiente onde um drone virtual poderá ser inserido. As máquinas virtuais de um determinado ambiente são criadas a partir de um dos Modelos de VMs cadastrados, são únicos e seguem o padrão "*<nome do ambiente>-vm<número identificador>*". Para o acesso a cada ambiente, é realizado um controle de acesso no nível de rede através de um filtro de pacotes (firewall) definido a partir de um perfil de firewall associado ao ambiente que é composto de um arquivo para carregamento (load) que é executado na inicialização do ambiente, outro para descarregamento (unload) que é executado na finalização de um ambiente, um nome e uma descrição (opcional). Um segundo controle deve ser realizado para restringir o acesso ao sistema a usuários que possuam um registro de usuário associado a um papel de segurança. Cada usuário deve ter um login único, um nome e e-mail único. Já os papéis de segurança devem ter um código de identificação único e um nome único.

¹⁰Este texto considera uma *Access Control List* (ACL) como a lista de permissões que um usuário para acesso a um determinado componente.

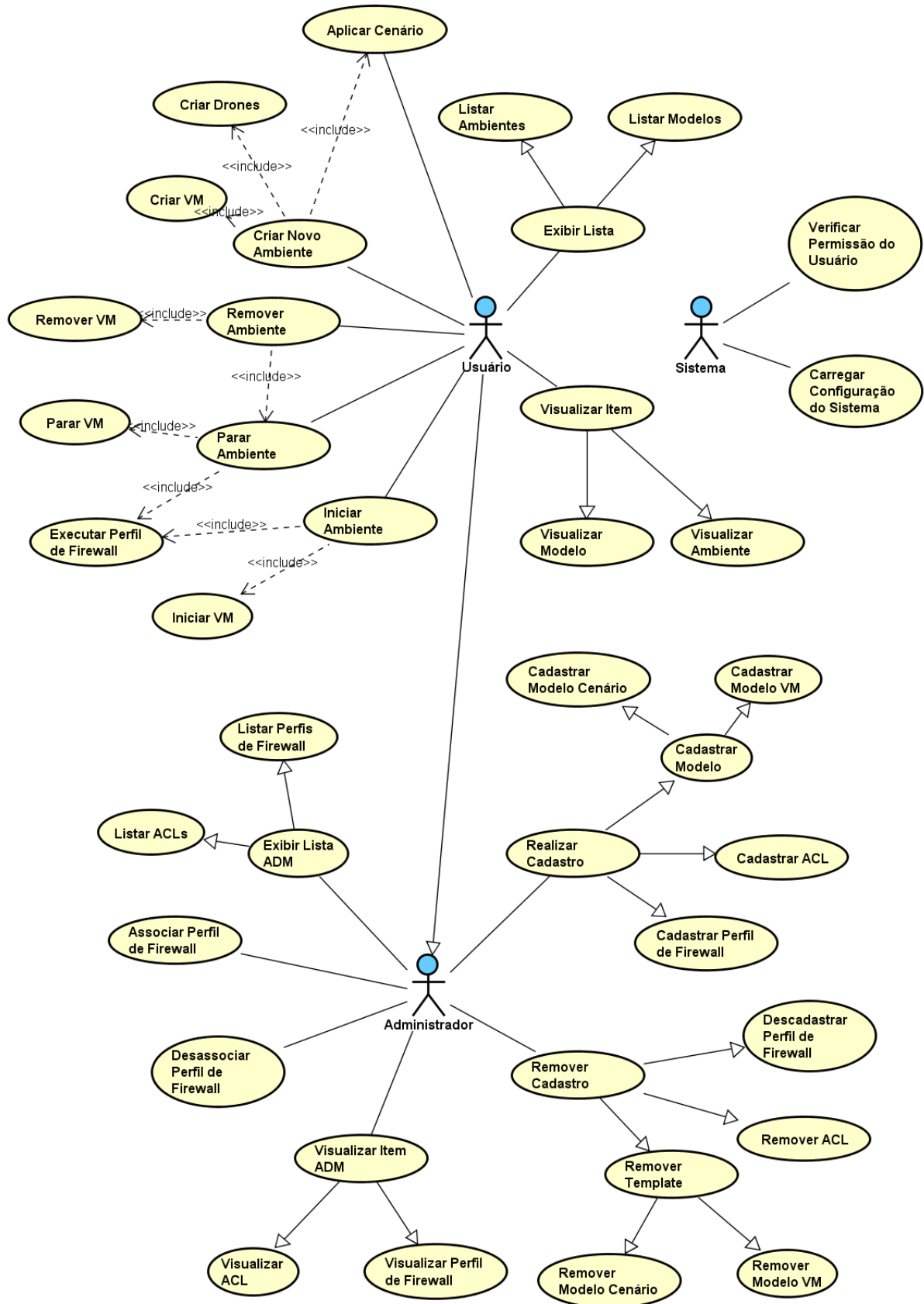


Figura 4.5: Diagrama de Casos de Uso do *rtcloud-cli*.
Fonte: Autor.

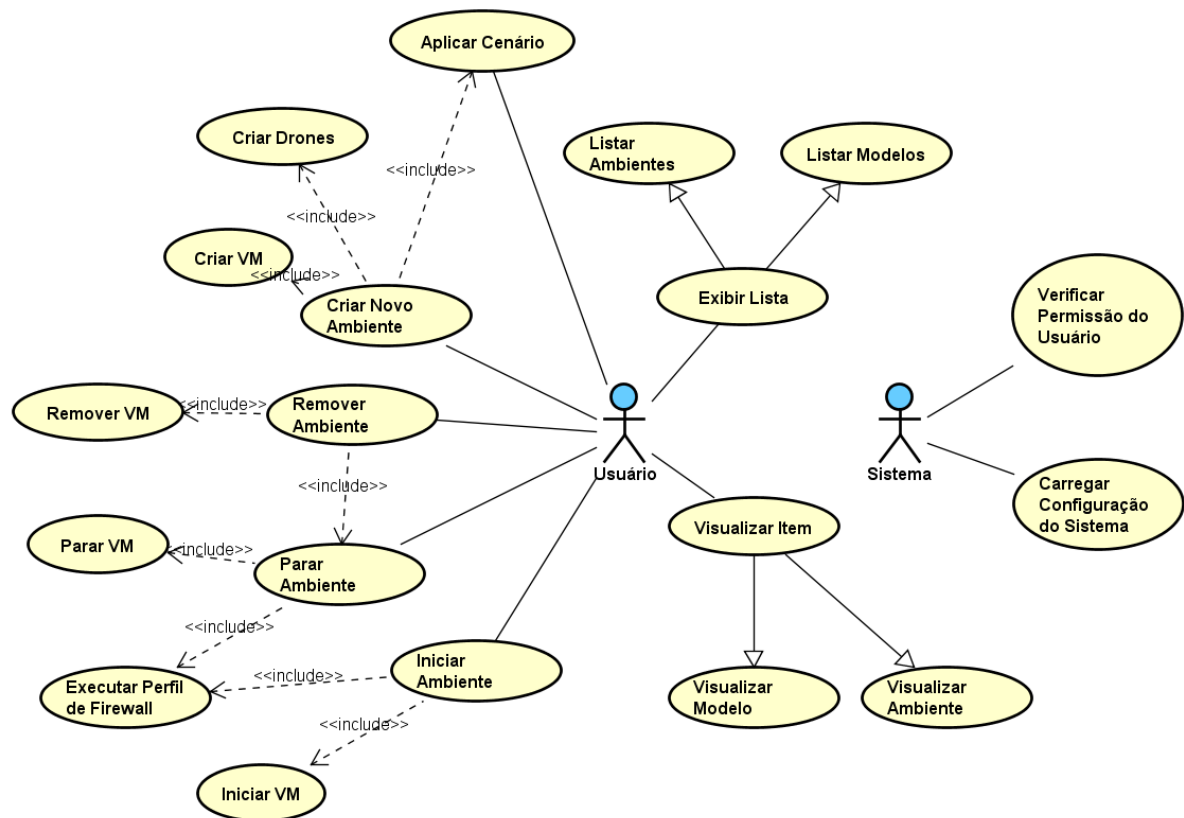


Figura 4.6: Diagrama de Casos de Uso do escopo definido para o *rtcloud-cli*.

Fonte: Autor.

Opcionalmente, um papel de segurança pode ter uma descrição e ser associado a um conjunto de ACLs onde cada ACL é composta de um código de identificação único, um ambiente robótico associado, uma operação permitida para este ambiente e uma descrição opcional. Uma operação só poderá ser realizada se o usuário estiver associado a um papel de segurança que contenha uma ACL permitindo a realização desta operação.

Os requisitos não funcionais foram agrupados unicamente no nível do Sistema. Sendo compatível com sistemas operacionais Windows e Linux, deverá realizar o gerenciamento remoto dos ambientes através de um canal de comunicação seguro. O usuário considerado para as operações do sistema e utilizado na Camada de Validação será o mesmo autenticado pelo Sistema Operacional. As tabelas 4.1 e 4.2 resumem os requisitos funcionais e não funcionais, respectivamente.

4.4.1.2 Diagrama de Classes Após o processo de elicitação de requisitos e construção dos diagramas de Casos de Uso e Sequência foi iniciada a criação do Diagrama de Classes da aplicação. Suas classes serão descritas nas próximas seções, estando o diagrama completo disponível no Apêndice A.2. Baseado nas funcionalidades necessárias ao sistema foram inicialmente levantados os padrões de projetos que poderiam atender a

Tabela 4.1: Requisitos funcionais do sistema *rtcloud-cli*.

Nível	Descrição
Usuário	O usuário pode criar um ambiente robótico virtualizado composto por múltiplas VMs, cada uma com múltiplos drones virtuais, fornecendo nome do ambiente, quantidade de VMs, Modelo de VM a ser utilizado, Modelo de Cenário Robótico e quantidade de drones.
	Um usuário pode remover um ambiente de simulação.
	Um usuário pode inicializar ou parar um ambiente de simulação.
	Um usuário pode listar os ambientes disponíveis para ele.
	Um usuário poderá listar todos os modelos disponíveis para ele.
	Um administrador poderá realizar todas as operações de um usuário, além das operações de administração do ambiente.
	O administrador pode associar ou desassociar perfis de firewall aos ambientes de simulação.
	O administrador pode listar todos os perfis de firewall e ACLs disponíveis.
	O administrador pode criar e remover ACLs.
	O administrador pode criar e remover Modelos.
O administrador pode criar e remover perfis de firewall.	
Sistema	Um ambiente robótico é composto de uma VM, um modelo cenário robótico associado a ele e pelo menos um drone. Um ambiente possui uma identificação única, um nome único e seu número de drones.
	Um modelo de cenário robótico é composto de arquivos e configurações que definem em qual cenário o drone irá executar. Ex.: Ambiente aberto com edifícios.
	Um modelo de cenário robótico deve ser associado ao ambiente durante a sua criação.
	Um papel de segurança é composto por um código de identificação único, um nome único e zero ou mais ACLs. Opcionalmente pode ter uma descrição.
	Uma ACL é composta de um código de identificação único, uma operação e o nome de um ambiente. Opcionalmente pode ter uma descrição.
	Um usuário deve ter um nome, um e-mail único, um papel e um login único. Opcionalmente pode ter uma observação a seu respeito.
	Um usuário só pode utilizar o sistema se possuir um registro de usuário com papel de segurança registrado.
	Um usuário só pode realizar uma operação específica no sistema se tiver alguma ACL permitindo esta execução desta operação.
	O nome de uma VM é criado automaticamente pelo sistema no formato <nome do ambiente>-vm<numero identificador>. Este nome é único. Ex.: amb-teste01.
	Uma VM só pode ser criada a partir de um modelo (template).
	Um ambiente pode ter um perfil de firewall associado a ele. Um perfil de firewall é composto de um arquivo de carregamento (load), um arquivo de descarregamento (unload), um nome e opcionalmente uma descrição.
	Um perfil de firewall deve ser carregado toda vez que um ambiente é inicializado.
Um perfil de firewall deve ser descarregado toda vez que um ambiente é finalizado.	

Tabela 4.2: Requisitos não funcionais do sistema *rtcloud-cli*.

Nível	Descrição
Sistema	O sistema deverá ser executado nos sistemas operacionais Linux e Windows.
	O usuário considerado no sistema deverá ser o mesmo usuário autenticado no sistema operacional. Logo, o sistema operacional deverá ser configurado para permitir que a aplicação obter os dados deste usuário.
	O sistema deverá utilizar conexão criptografada para a execução de comandos remotos.
	O sistema deverá permitir o gerenciamento remoto dos ambientes, não sendo necessário estar instalado no domínio de controle da camada de virtualização.

estas funcionalidades. A utilização dos padrões foi adotada com o objetivo de diminuir a dependência entre classes, facilitar a manutenção e trazer maior flexibilidade ao código. Os padrões de projeto utilizados foram implementados com base em (Gamma, 1995; Freeman e Freeman, 2007) e serão descritos nas próximas seções. A Tabela 4.3 contém um resumo da utilização de cada padrão implementado. O padrão *Façade* foi utilizado para diminuir o acoplamento entre classes realizando a comunicação entre elas através das classes *API* (para comunicação externa) e *RTCloud* (para comunicação interna). O Padrão DAO foi utilizado para separar as regras de acesso ao banco das demais regras do sistema. Para simplificar a criação das classes DAO, o padrão *Factory* foi adotado. O padrão *Singleton* foi utilizado para garantir que algumas classes tivessem apenas uma instância durante toda a execução da aplicação. Por fim, o padrão *Command* foi utilizado para encapsular em objetos as solicitações dos usuários, facilitando seu tratamento pela API de Gerenciamento.

Tabela 4.3: Padrões de projeto adotados no *rtcloud-cli*.

Padrão	Aplicação
Façade	Fachada principal do sistema e API de Gerenciamento.
Data Access Object (DAO)	Acesso à camada de persistência dos dados.
Singleton	Fachada principal do sistema, classe para criação de DAOs e classe de parâmetros do sistema.
Factory	Criação de DAOs.
Command	Interface de linha de comando.

Além dos padrões de projeto citados, também foi adotado o Controle de Acesso Baseado em Papéis, ou *Role Based Access Control* (RBAC) em inglês, uma abordagem baseada na associação de papéis que reúnem um conjunto de permissões a objetos e usuários. Esta implementação será descrita na Subseção 4.4.5.

A configuração da aplicação é realizada criando um arquivo de parâmetros em formato texto com linhas no formato *parametro=valor* conforme exemplo da Figura 4.7. Na inicialização do *rtcloud-cli* este arquivo é lido e seu conteúdo é armazenado em uma

instância da classe *SystemParameters*, implementada utilizando o padrão *Singleton*. A instanciação desta classe já é suficiente para a realização da carga do arquivo, contudo, é possível posteriormente atualizar estes valores através do método *loadParam()* da classe *RTCloud*. Desta forma, demais classes não precisam realizar o acesso direto à classe *SystemParameters*, diminuindo assim seu acoplamento. O relacionamento entre estas classes pode ser visto no Diagrama de Classes da Figura 4.8. O nome padrão para o arquivo de parâmetros é *rtcloud.conf*, contudo é possível especificar o caminho completo do arquivo de parâmetros adicionando o parâmetro *--pfile=<filepath>* à linha de comando do *rtcloud-cli*.

```
D:\OneDrive - Stanly Community College\Repositorios\Git\tcc\src>type rtcloud.conf
debug=False
dbName=rtclouddb
dbHost=localhost
dbUser=rtcloud
dbPassword=Cloud2016UAV#
hypervisorIp=10.0.5.40
hypervisorLogin=root
hypervisorSSHPort=2131
hypervisorPasswd=Cloud2016UAV#
defaultVMPPath=/var/domains/01/
defaultTemplate=template-default
defaultScenario=template-scenario-default
globalFwProfile='/usr/local/scripts/rtcloud/fwprofile.global.sh'
defaultFwProfileId=1
virtClonePath=/usr/bin/virt-clone
virtCustomizePath=/usr/bin/virt-customize
D:\OneDrive - Stanly Community College\Repositorios\Git\tcc\src>
```

Figura 4.7: *rtcloud-cli* - Exemplo de arquivo de parâmetros.
Fonte: Autor.

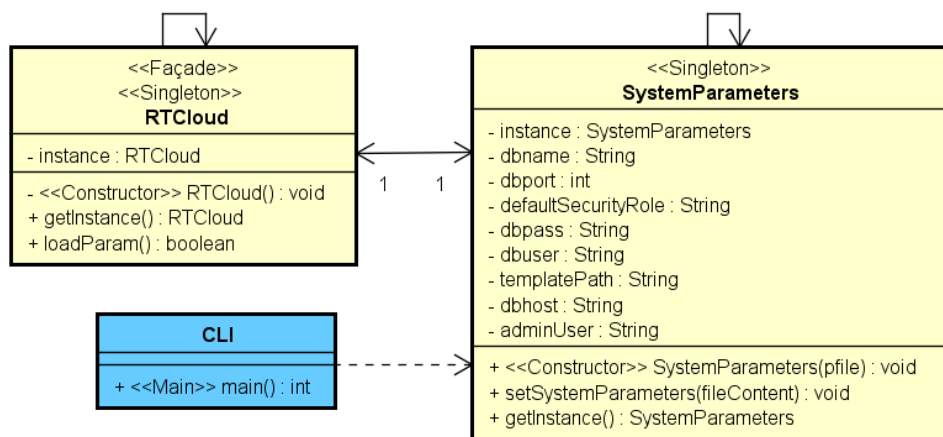


Figura 4.8: Diagrama de Classes - Carregamento do arquivo de parâmetros.
Fonte: Autor.

4.4.2 Interface do Usuário e API de Gerenciamento de Ambientes

A interface do usuário é baseada em linha de comando sendo suportada nos sistemas operacionais Windows e Linux. Cada execução do sistema realiza uma única tarefa solicitada pelo usuário através da especificação de um comando e um conjunto de parâmetros auxiliares que fornecem as informações necessárias para a aplicação realizar a execução da tarefa. O tratamento destes comandos é realizado através do padrão de projeto *Command*, como visto na Figura 4.9.

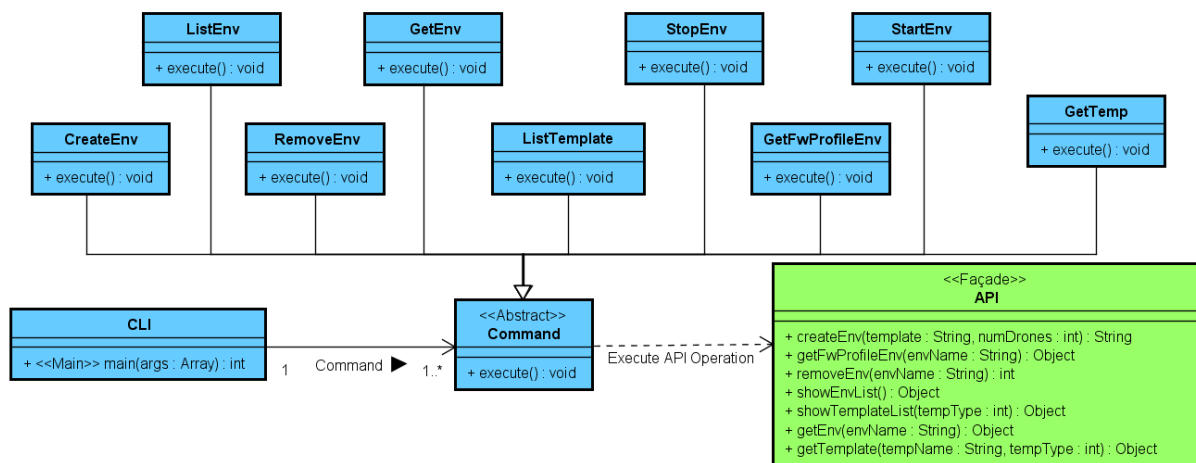


Figura 4.9: Diagrama de Classes - Interface do Usuário e API de Gerenciamento.
Fonte: Autor.

Este padrão verifica se o comando solicitado está na lista de comandos aceitos e, em caso positivo, utiliza os métodos disponibilizados na API de Gerenciamento para atender às solicitações do usuário. O comando a ser executado é selecionado a partir de uma estrutura de dados tipo *hashmap* que assume o papel de Invocador (*Invoker* em inglês) nesta implementação, evitando assim a necessidade de utilização de estruturas de seleção como *if-elseif-else* ou *switch-case*. Caso o comando não exista, é exibida uma mensagem de erro na saída de erro padrão (*stderr*) e o fluxo de execução é encerrado sendo emitido para o Sistema Operacional um código de retorno. A Tabela 4.4 relaciona os comandos aceitos pelo *rtcloud-cli*, as classes de comando do padrão *Command* implementado e os respectivos métodos invocados na API.

A API de Gerenciamento de Ambientes foi implementada utilizando o padrão de projeto *Façade* (Gamma, 1995) e assume o papel de Receptor (*Receiver* em inglês) do padrão *Command* (Gamma, 1995). Neste padrão, o papel do receptor é receber as requisições de cada classe de comando. A API realiza o tratamento das requisições realizadas pela Interface do Usuário e interage com a classe *RTCloud*, a fachada principal do sistema, para realizar as operações necessárias para o atendimento da requisição do usuário. Seu diagrama pode ser visto na Figura 4.9.

As Figuras 4.10 e 4.11 exemplificam a utilização do *rtcloudcli* com os comandos *listEnv* e *showEnv*, respectivamente. A primeira mostra a execução do comando *listEnv*

Tabela 4.4: Descrição de comandos aceitos pelo sistema e suas respectivas classes de comando.

Comando	Classe de Comando	Método da API	Descrição
listEnv	ListEnv	listEnv()	Exibe a lista dos ambientes disponíveis para o usuário.
showEnv	GetEnv	getEnvByName()	Exibe informações sobre um ambiente especificado pelo usuário através do parâmetro <code>--envName</code> .
createEnv	CreateEnv	createEnv()	Cria um novo ambiente a partir de um modelo de VM disponível no sistema. O usuário especifica um nome para o novo ambiente (<code>--envName</code>), um modelo (<code>--tempName</code>) e um perfil de firewall (<code>--fwProfile</code>) para o novo ambiente.
removeEnv	RemoveEnv	removeEnv()	Remove um ambiente especificado pelo usuário através do parâmetro <code>--envName</code> .
listTemplates	ListTemplate	listTemp()	Exibe a lista dos modelos disponíveis para o usuário.
showTemplate	GetTemp	getTempByName()	Exibe informações sobre um modelo especificado pelo usuário através do parâmetro <code>--tempName</code> .
getFwProfile	GetFwProfileEnv	GetFwProfByName()	Exibe informações sobre um perfil de firewall associado ao ambiente especificado pelo usuário através do parâmetro <code>--envName</code> .
startEnv	StartEnv	startEnv()	Inicializa um ambiente especificado pelo usuário através do parâmetro <code>--envName</code> .
stopEnv	StopEnv	stopEnv()	Para um ambiente especificado pelo usuário através do parâmetro <code>--envName</code> .
applyScenario	ApplyScenario	applyScenario()	Associa um modelo de cenário (<code>--scenarioName</code>) a um ambiente (<code>--envName</code>).

resultando na lista dos ambientes atualmente registrados. A segunda exibe os detalhes de um ambiente solicitado através do comando `showEnv`.

```
D:\OneDrive - Stanly Community College\Repositorios\Git\tcc\src>rtcloudcli.py listEnv
System parameter file rtcloud.conf loaded with 16 parameters.
Getting environment list...
Found 3 environments:

1: Default
2: all
3: env-alexandre

D:\OneDrive - Stanly Community College\Repositorios\Git\tcc\src>
```

Figura 4.10: Exemplo de uso do rtcloudcli: Lista de ambientes.

Fonte: Autor.

```
D:\OneDrive - Stanly Community College\Repositorios\Git\tcc\src>rtcloudcli.py showEnv --envName=env-alexandre
System parameter file rtcloud.conf loaded with 16 parameters.
Getting environment env-alexandre...
Environment Id: 143
Environment Name: env-alexandre
Environment Template Id: 1
Environment Firewall Profile Id: 1
Environment VM path: /var/domains/01/
Environment number of drones: 1
Environment VM IP: 192.168.122.4

D:\OneDrive - Stanly Community College\Repositorios\Git\tcc\src>
```

Figura 4.11: Exemplo de uso do rtcloudcli: Exibe informações de um ambiente.

Fonte: Autor.

4.4.3 Camada de Gerenciamento

A Camada de Gerenciamento foi implementada na aplicação `rtcloud-cli` e é responsável por orquestrar a execução das tarefas necessárias para a realização dos Casos de Uso da

Figura 4.6 que estão relacionadas com o usuário. As quatro classes que compõem esta camada podem ser vistas em amarelo na Figura 4.12 e as classes pertencentes a outras camadas estão marcadas em verde. A principal classe é a *RTCloud*, pois recebe todas as solicitações destinadas a esta camada e orquestra a sua realização invocando métodos das demais classes. A classe *RoboticEnvironment* é a segunda mais importante, pois realiza as operações nos ambientes e nas máquinas virtuais através da classe *VirtualMachine*. A classe *OSOperation* dá apoio às demais classes realizando a execução de operações no nível do sistema operacional como a leitura de arquivos e acesso remoto ao Domínio de Controle (Dom0).

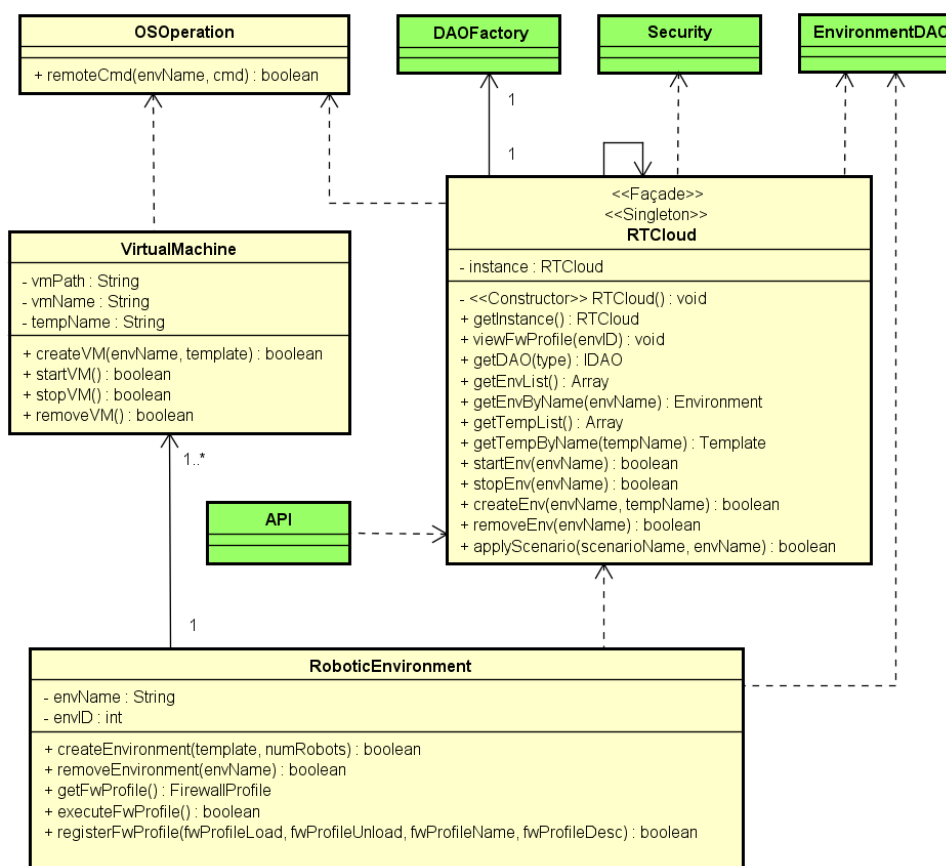


Figura 4.12: Diagrama de Classes - Camada de Gerenciamento.

Fonte: Autor.

A execução remota de comandos no Domínio de Controle ocorre através de um túnel criptografado criado pelo protocolo SSH versão 2 disponibilizado pela biblioteca *Paramiko*¹¹. A Camada de Gerenciamento faz o acesso remoto ao Domínio de Controle para executar os seguintes comandos:

- a) *virsh*: Utilizado para o gerenciamento de máquinas virtuais.

¹¹Disponível em: (<http://www.paramiko.org/>). Acessado em 29/07/2017.

- b) *virt-clone*: Comando baseado na biblioteca *libvirt* utilizada para criar uma máquina virtual a partir de um modelo. Nesta implementação, um Modelo de VM é uma máquina virtual que não está associada a um ambiente de simulação, mas está registrada no sistema como Modelo de VM.
- c) *virt-customize*: Comando que executa operações de personalização da máquina virtual como troca do *hostname*, cópia e remoção de arquivos, instalação de pacotes do Sistema Operacional e execução de scripts, dentre outros.
- d) *apply_scenario.sh*¹²: Este script realiza a instalação de um cenário em um determinado domínio. O processo de instalação consiste em copiar o conteúdo do diretório do cenário para o diretório */var/tmp* do sistema de arquivos instalado no domínio e executar o script *scenario.install.sh* contido no diretório do cenário. Este script deverá realizar as operações de instalação do cenário, como cópia de arquivos, compilação de binários e instalação de pacotes do SO, dentre outros.

Dentro do processo de criação de um novo ambiente a Camada de Gerenciamento realiza a personalização de um domínio recém criado executando os seguintes passos com a ferramenta *virt-customize*:

- a) Configuração do nome de rede (*hostname*) para o novo domínio;
- b) Configuração do endereço IP e demais parâmetros de rede, realizado através da execução do script *setup_interface_debian.sh*¹³ no SO do novo domínio.
- c) Troca das chaves para o SSH, realizado através da execução do script *change_ssh_key.sh* no SO do novo domínio.
- d) Execução de script de personalização *custom_user.sh* no SO do Domínio de Controle. Este script permite que o administrador possa adicionar comandos para a personalização do novo domínio sem a necessidade de alteração no código do *rtcloud-cli*, podendo inclusive utilizar o *virt-customize* para realizar outras operações dentro do novo domínio.

Como pôde ser visto, durante a personalização de um novo ambiente está o passo de configuração do endereço IP para a nova máquina virtual. Este endereço é o primeiro IP não utilizado dentre um conjunto de configurações de IPs que são registrados na tabela *ip_pool* a ser descrita na Subseção 4.4.4. Ao utilizar um IP o sistema marca-o como em uso, sendo marcado novamente como não utilizado apenas quando o respectivo ambiente é removido.

As figuras 4.13 e 4.14 demonstram a inicialização de um ambiente utilizando o *rtcloud-cli* e a verificação da máquina virtual em execução na Camada de Virtualização, respectivamente. Já as figuras 4.15 e 4.16 demonstram a parada deste ambiente e a verificação da máquina virtual fora de execução, respectivamente.

¹²Os scripts utilizados nesta implementação localizam-se em */usr/local/scripts*

¹³Todos os scripts de personalização estão localizados em */usr/local/scripts/customize_domain/*.

```
D:\OneDrive - Stanly Community College\Repositorios\Git\tcc\src>rtcloudcli.py startEnv --envName=env-alexandre
System parameter file rtcloud.conf loaded with 16 parameters.
Starting environment env-alexandre...
VM started.
D:\OneDrive - Stanly Community College\Repositorios\Git\tcc\src>
```

Figura 4.13: Exemplo de uso do rtcloudcli: Inicialização de um ambiente.

Fonte: Autor.

```
[root@harris ~]# virsh list | grep alexandre
84      env-alexandre00      running
[root@harris ~]#
```

Figura 4.14: VM executando após inicialização.

Fonte: Autor.

```
D:\OneDrive - Stanly Community College\Repositorios\Git\tcc\src>rtcloudcli.py stopEnv --envName=env-alexandre
System parameter file rtcloud.conf loaded with 16 parameters.
Stopping environment env-alexandre...
VM Stopped.
D:\OneDrive - Stanly Community College\Repositorios\Git\tcc\src>
```

Figura 4.15: Exemplo de uso do rtcloudcli: Parada de um ambiente.

Fonte: Autor.

```
[root@harris ~]# virsh list | grep alexandre
84      env-alexandre00      shut off
[root@harris ~]# virsh list | grep alexandre
[root@harris ~]#
```

Figura 4.16: VM não executando após parada do ambiente.

Fonte: Autor.

4.4.4 Camada de Persistência de dados

A camada de persistência de dados foi implementada com o banco de dados MySQL versão 5.7 (Oracle Corporation., 2017) utilizando tabelas armazenadas sob o mecanismo Innodb (Oracle Corporation, 2017), que garante as restrições no uso de chaves estrangeiras. O modelo de dados desenhado para o sistema é composto de oito tabelas de dados que podem ser vistas na Figura 4.17. A tabela *environment* contém as informações referentes aos ambientes disponíveis para os pesquisadores sendo a tabela principal do sistema e que possui mais relacionamentos com as demais. A tabela *template* contém as informações referentes aos modelos de VMs e Cenários disponíveis no sistema. O atributo *tipo* pode ter como conteúdo as strings "vm" ou "cenario" e define a qual tipo de modelo aquela entrada se refere. A tabela *ip_pool* mantém o estado do conjunto de endereços IPs disponíveis para a utilização das VMs que compõem os ambientes. O atributo booleano *used* indica se o endereço está em uso ou disponível para ser atribuído a uma nova VM. Já as tabelas *role*, *acl* e *user* são utilizadas pela Camada de Validação de Segurança descrita na Subseção 4.4.5 para persistência dos dados referentes aos papéis de usuários, listas de acesso e usuários, respectivamente, sendo que o tipo de relacionamento muitos-para-muitos entre as tabelas *role* e *acl* é implementado utilizando-se a tabela auxiliar *role_acl*. Por fim, temos a tabela *fw_profile* que contém os *Perfis de Firewall* que

fazem parte da implementação da Camada de Filtro de Segurança descrita posteriormente na 4.4.8.

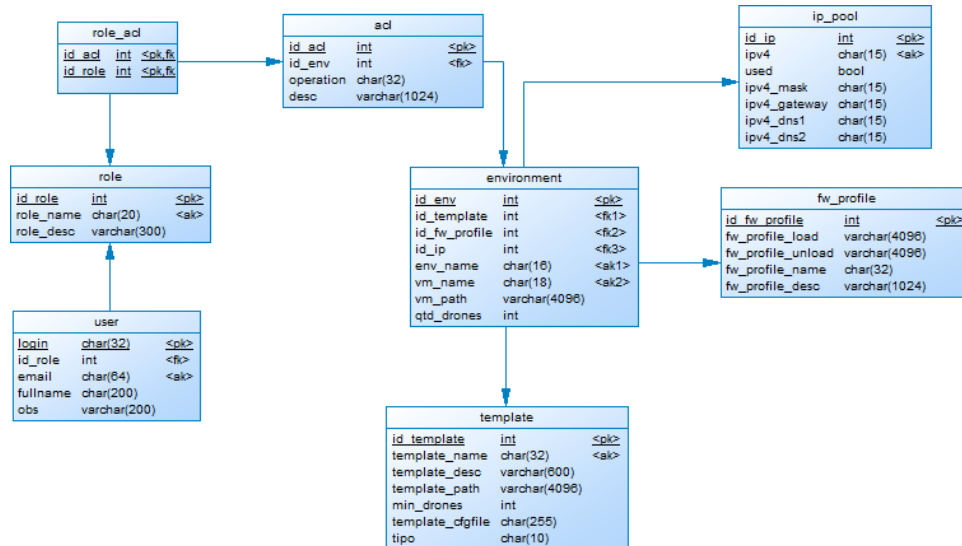


Figura 4.17: Modelo físico do banco de dados.

Fonte: Autor.

O acesso à Camada de Persistência é realizado pelas camadas de Interface do Usuário (descrita na Subseção 4.4.1), Gerenciamento (descrita na Subseção 4.4.3) e Validação de Segurança (descrita na Subseção 4.4.1) implementados na aplicação *rtcloud-cli*. A Figura 4.18 mostra o diagrama de classes desta implementação que é fundamentada nos padrões de projeto Data Access Object (DAO) (Gamma, 1995), utilizado para a manipulação direta dos dados, e Factory Method (Gamma, 1995), utilizado para a criação dos objetos de acesso aos dados. O padrão DAO implementado é composto da *interface* chamada *IDAO*, três classes especializadas e suas respectivas classes para os objetos de transferência de dados (Transfer Objects):

- EnvironmentDAO: É responsável pela manipulação das informações referentes aos ambientes contidas nas tabelas *environment*, *ip_pool* e *fw_profile*. Utiliza a classe *Environment* para os objetos de transferência de dados;
- TemplateDAO: É responsável pela manipulação das informações referentes aos modelos contidas na tabela *template*. Utiliza a classe *Template* para os objetos de transferência de dados;
- SecurityDAO: É responsável pela manipulação das informações referentes aos modelos contidas nas tabelas *user*, *role*, *acl* e *role_acl*. Utiliza a classe *SecurityRole* para os objetos de transferência de dados.

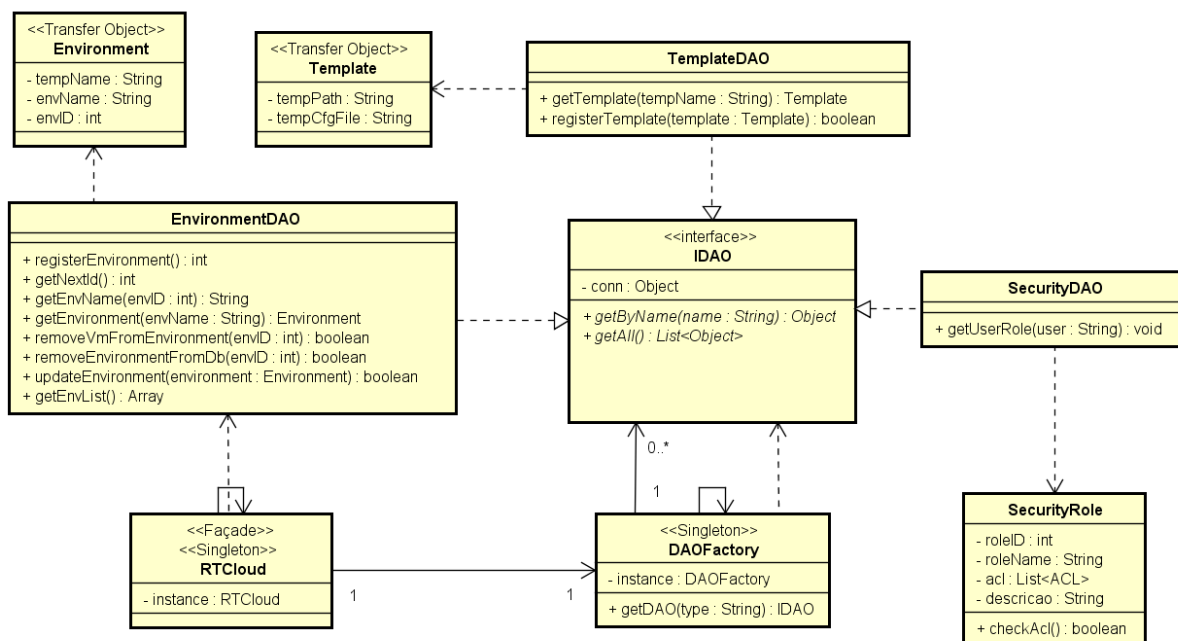


Figura 4.18: Diagrama de Classes - Camada de Persistência de Dados.

Fonte: Autor.

O padrão Factory Method (Gamma, 1995) é implementado através da tabela *DAOFactory* que concentra a criação de instâncias das classes especializadas de acordo com o tipo de classe DAO especificado como atributo de entrada na chamada do método *getDAO()* que, por sua vez, retorna um objeto do tipo *IDAO* apontando para a instância de classe especializada criada.

4.4.5 Camada de Validação de Segurança

A Camada de Validação de Segurança foi implementada com base no modelo RBAC básico (Sandhu et al., 2000) descrito pelo NIST¹⁴. O modelo RBAC regula o acesso de usuários a objetos através de papéis funcionais que possuem as permissões que habilitam a execução de operações nestes objetos. A gerência de permissões neste modelo é simplificada consideravelmente, uma vez que não é necessário definir e gerenciar as permissões para cada usuário. Esta tarefa é realizada unicamente nos papéis e todos os usuários associados a estes serão afetados pelas mudanças e, como nas organizações é mais fácil existir a troca de membros do que alterações nas funções, o trabalho repetido de gerenciar as mesmas permissões para cada usuário é eliminado. A flexibilidade também é uma característica interessante para o sistema implementado, pois o modelo pode evoluir de uma implementação bem simples, como foi implementado aqui, até estruturas mais complexa baseada em hierarquias a depender das necessidades futuras do LaSiD (Sandhu et al., 2000). A Figura 4.19 mostra a estrutura básica definida pelo NIST.

¹⁴National Institute of Standards and Technology (NIST)

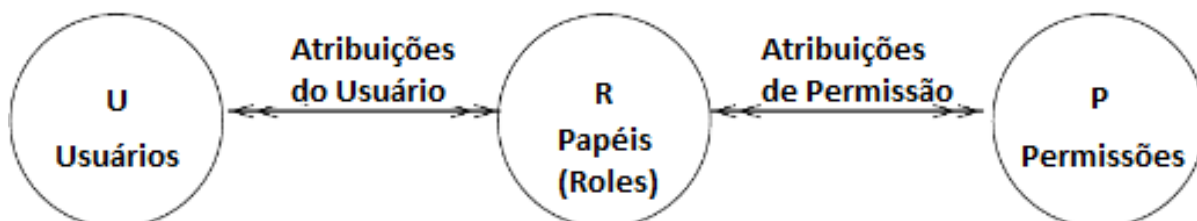


Figura 4.19: Diagrama do modelo RBAC Básico do NIST.

Fonte: Baseado em (Sandhu et al., 2000)

A verificação de autorização para execução das operações solicitadas pelo usuário é invocada pela classe de fachada principal *RTCloud* em todos os métodos utilizados pela API. A decisão de utilizar esta classe como ponto de verificação foi tomada com objetivo de diminuir o acoplamento entre as classes, permitindo que a API possa ser implementada em uma aplicação externa ao *rtcloud-cli* com mais facilidade caso futuramente seja necessário haver esta separação. A gerência de sessões, que inclui autenticação do usuário, foi delegada ao sistema operacional, uma vez que a aplicação utiliza os dados do usuário que a está executando para realizar o processo de autorização. Desta forma, um pré-requisito para sua utilização é que o sistema operacional possa informar qual usuário está autenticado naquela sessão e que esta informação possa ser localizada na base de dados do *rtcloud-cli*. A Figura 4.20 mostra o Diagrama de Classes da implementação do modelo RBAC realizada neste trabalho.

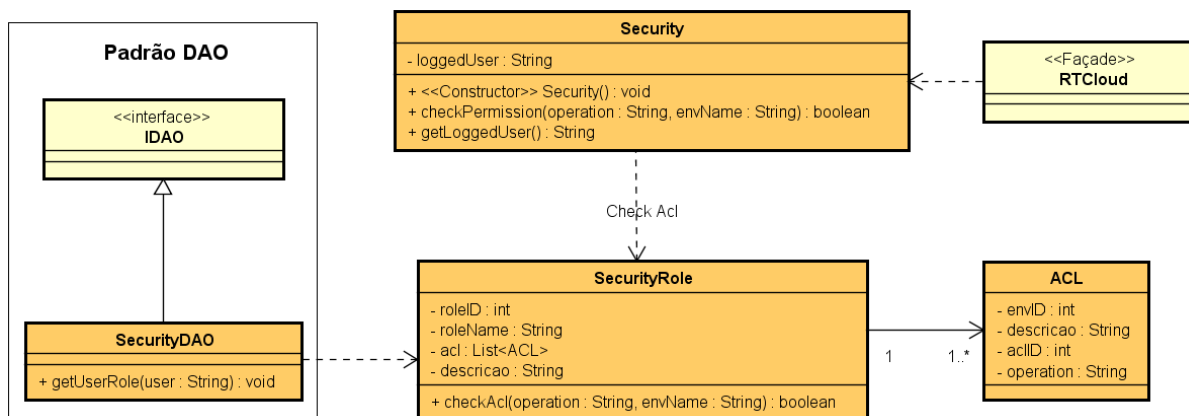


Figura 4.20: Diagrama de Classes do modelo RBAC implementado.

Fonte: Autor.

4.4.6 Camada de Virtualização

Para a escolha do sistema de virtualização a ser utilizado na Nuvem de Simulação do LaSiD foi realizada uma busca em pesquisas referentes a virtualização de aplicações com requisitos de tempo real e nuvens robóticas com o objetivo de levantar soluções que fossem aderentes aos projetos de pesquisa acadêmica atualmente desenvolvidos pelo laboratório. Esta pesquisa levou em consideração aspectos relevantes para o desenvolvimento

acadêmico, dentre eles:

- a) Utilização na área acadêmica;
- b) Aderência a equipamentos acessíveis ao LaSiD;
- c) Estado de atividade do projeto;
- d) Possibilidade de implementação;
- e) Integração com ferramentas de terceiros.

Um ponto relevante para a escolha do virtualizador é o aproveitamento destas pesquisas acadêmicas para o seu desenvolvimento. O RT-Xen é um exemplo, pois foi incorporado ao código do Xen versão 4.5 e desde então vem sofrendo melhorias. Isto traz uma consequência positiva para o presente estudo: suporte oficial para a plataforma ARM, uma vez que o Xen já suporta esta plataforma, além da plataforma x86. Um ponto desejável para a camada de virtualização citada na [Subseção 4.3.7](#) é a possibilidade do virtualizador poder ser utilizado também em sistemas embarcados, sendo os processadores ARM amplamente utilizados nestes sistemas, a exemplo dos quadrotoros existentes no LaSiD (ver [Subseção 3.2.5](#)). Além da plataforma ARM, é essencial o suporte à plataforma x86 utilizada atualmente no LaSiD. Por último, o Xen possui integração com muitas ferramentas de terceiros que auxiliam na sua administração, como o próprio *libvirt*, e que fazem gerência de nuvem, como o OpenStack, CloudStack, Eucalyptus, dentre outros.

Em relação ao sistema operacional utilizado nos domínios, o GNU/Linux foi escolhido após a análise dos artigos, teses e sites utilizados neste estudo indicar uma maior aderência das tecnologias envolvidas a este sistema operacional. O Domínio de Controle tem a distribuição Fedora Server release 24 instalada¹⁵. Os demais domínios rodam sobre a distribuição Ubuntu 14.04.4 LTS por ser oficialmente suportado pelo ROS versão *Indigo*, utilizada no presente estudo¹⁶. Neste estudo foi utilizada a paravirtualização, pois esta diminui a sobrecarga de recursos imposta pela emulação de recursos do sistema ([Xen Project Community, 2016](#)).

4.4.7 Camada de simulação

A camada de simulação disponibiliza uma plataforma de programação para aplicações robóticas. Uma instância desta camada é uma máquina virtual que abriga, além do sistema operacional, dois componentes principais que estão integrados:

- a) Robot Operating System versão *Indigo* (ver [Seção 2.4](#));
- b) Gazebo versão 2.2.3 (ver [Seção 2.5](#)) e seu pacote de integração com o ROS *Indigo*;

¹⁵Instruções de instalação específica para o Xen disponível em:

<https://wiki.xen.org/wiki/Fedora.Host.Installation>

<https://fedoraproject.org/wiki/Archive:Tools/Xen?rd=FedoraXenQuickstart>

¹⁶Instruções de instalação indicando o suporte à distribuição Ubuntu disponível em <http://wiki.ros.org/indigo/Installation>

Apesar desta combinação permitir a programação baseada em diversos tipos de robôs simulados o presente trabalho teve como objetivo entregar um ambiente com o quadrotor AR.Drone simulado. Sendo assim, foi adicionado aos componentes já citados o Simulador do quadrotor Parrot AR.Drone descrito na [Seção 3.3](#), sendo utilizada uma versão adaptada para o ROS versão Indigo, disponível em ([Kohorn, 2015](#)), uma vez que a versão indicada na página oficial é muito antiga, utiliza componentes já depreciados nas versões mais novas do ROS e, com isto, dificulta futuras melhorias. Contudo, com esta adaptação não houveram mudanças significativas na arquitetura do pacote.

A instalação do ROS, Gazebo e o pacote de integração entre eles foi realizada utilizando os pacotes disponíveis no sistema de gerenciamento de pacotes do Ubuntu. Já o `tum_simulator` não disponibiliza pacote para esta distribuição, sendo realizada a instalação conforme indicado no site do desenvolvedor ([Kohorn, 2015](#)).

Os usuários têm acesso a esta camada através dos serviços de SSH e SFTP disponibilizados pelo Sistema Operacional sem limite de logins simultâneos possibilitando que um ou mais pesquisadores possam trabalhar no ambiente. Através destes meios é possível o pesquisador coletar dados referente ao experimento e exportar sessões de um ambiente gráfico para visualização dos cenários simulados. Como exemplo da utilização do ambiente a seguinte sequência de comandos foi executada na linha de comando do Sistema Operacional:

```
# Carrega o ambiente operacional para programação e simulação robótica.
source ~/tum_simulator_ws/deve/setup.sh

# Inicializa o Gazebo e o simulador do AR.Drone
roslaunch cvg_sim_gazebo ardrone_testworld.launch gui:=false headless:=true &

# Inicializa a visualização das câmeras frontal e inferior do drone simulado.
roslaunch image_view image_view image:=/ardrone/front/image_raw &
roslaunch image_view image_view image:=/ardrone/bottom/image_raw &

# Sequência de comandos enviados ao drone.
# Envia comando para decolagem.
rostopic pub -1 /ardrone/takeoff std_msgs/Empty

# Ir em frente.
rostopic pub -r 10 /cmd_vel geometry_msgs/Twist '{linear: {x: 1.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}'

# Aumentar a altitude.
rostopic pub -r 10 /cmd_vel geometry_msgs/Twist '{linear: {x: 0.0, y: 0.0, z: 1.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}'

# Realizar pouso.
rostopic pub -1 /ardrone/land std_msgs/Empty
```

As operações enviadas ao drone virtual foram executadas corretamente, contudo, o desempenho observado no ambiente de simulação esteve abaixo do esperado, impactando em atrasos na atualização dos quadros exibidos pelo simulador. Investigando os índices do sistema operacional, foi identificado um alto consumo dos processadores e carga (load average) alta, conforme pode ser visto na [Figura 4.21](#), sendo este comportamento aceitável uma vez que o hardware disponível para os experimentos não possui processadores gráficos capazes de apoiar a construção dos cenários da simulação. Ainda, as mesmas operações foram executadas acessando o ambiente via rede local e via VPN, notando-se um desempenho consideravelmente melhor na rede local. Via VPN foram testadas configurações com um, quatro e oito processadores virtuais, mas nenhuma delas forneceu um desempenho aceitável para a realização das operações. Já via rede local este desempenho foi alcançado na configuração de quatro processadores virtuais, mesmo com média de carga alta. A [Figura 4.22](#) mostra a execução dos comandos, com a visão das câmeras frontal e inferior e o panorama geral do cenário. Atualmente, uma sessão do ambiente de exibição gráfica X Windows Systems (ver [The Open Group \(2017\)](#)) é enviada via rede a um servidor X Windows localizado na estação do pesquisador, havendo assim a possibilidade de melhora nos resultados ao atacar outros protocolos de exibição gráfica via rede.

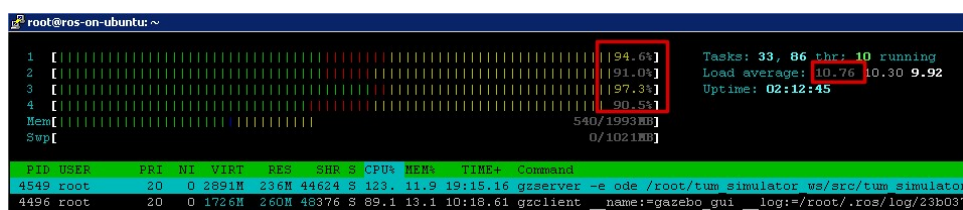


Figura 4.21: Utilização de processadores durante a simulação.

Fonte: Autor.

4.4.8 Filtro de Segurança

O filtro de segurança foi implementado usando o framework *IPFilter*. A ferramenta em linha de comando *IPTables* foi utilizada para configurar regras de filtragem de pacotes e tradução de endereços de rede (NAT, Network Address Translation). Um script com comandos *IPTables* foi criado contendo regras genéricas que devem ser executadas para cada máquina virtual iniciada no ambiente. Um exemplo de utilização deste script é permitir acesso à porta SSH dos domínios, uma vez que esta porta é padrão para todos. Este script é configurado pelo administrador do sistema dentro do arquivo de configuração do *rtcloud-cli*. Além deste, um script individual é definido na criação do ambiente para suprir necessidades individuais do ambiente. O *rtcloud-cli* relaciona os scripts individuais com seus respectivos ambientes através da camada de persistência de dados, sendo ele também o responsável por executá-los no momento em que o usuário solicita que o ambiente seja iniciado. O filtro localiza-se no Domínio de Controle, uma vez que as interfaces que recebem todo o tráfego para os domínios estão aí localizadas. A topologia de rede é descrita com mais detalhes a seguir na [Subseção 4.4.9](#).

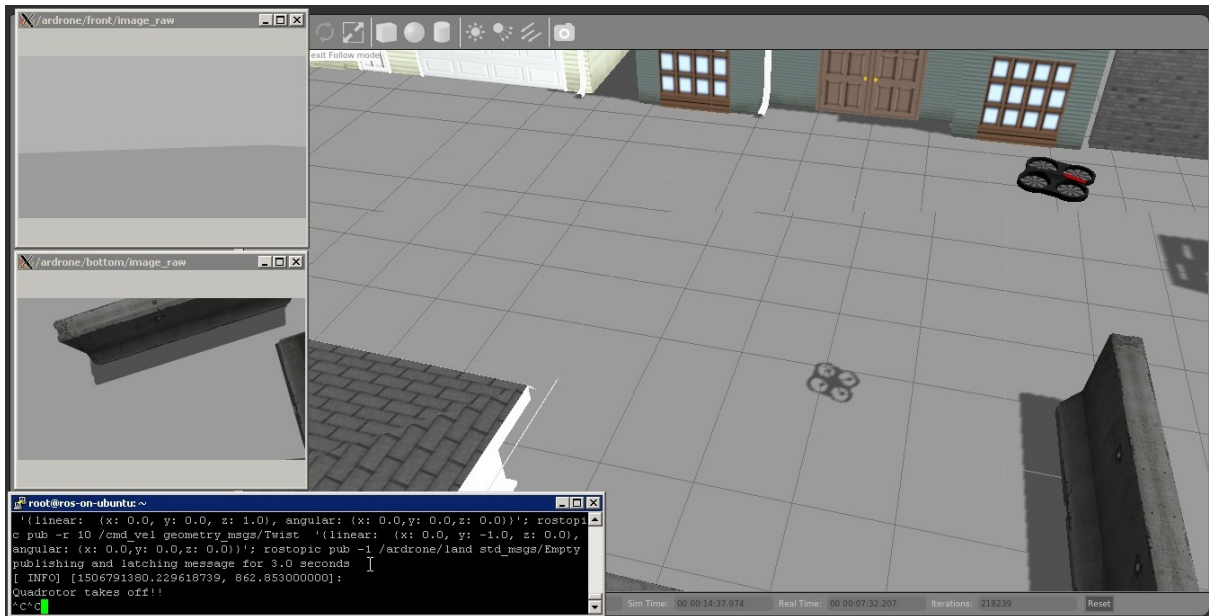


Figura 4.22: Cenário simulado e imagens das câmeras durante a simulação.

Fonte: Autor.

4.4.9 Rede

A Figura 4.23 descreve a topologia de rede baseada em pontes (Network Bridge)¹⁷ utilizada na implementação da Camada de Virtualização. Alguns fatores levados em consideração na escolha da topologia baseada em pontes foram:

- a) A topologia baseada em pontes permite que os domínios acessem diretamente a rede. Incluindo a interface física de acesso ao switch externo à configuração da ponte é possível permitir que os pacotes sejam copiados diretamente para esta interface. Uma vez que o acesso a rede é um desafio na pesquisa de robôs autônomos, para alguns experimentos este acesso direto pode vir a ser necessário;
- b) Ao contrário da topologia baseada em roteamento, não é necessário realizar configurações que referenciem o endereço IP do domínio, o que facilita a criação automatizada dos ambientes de simulação;
- c) Sua implementação é simples e a curva de aprendizado para sua manutenção é baixa, facilitando a manutenção por futuros mantenedores da nuvem.

Apesar da utilização de uma bridge para conexão dos domínios, estes são configurados para utilizar o domínio de controle (Dom0) como rota padrão, sendo então aí aplicados os filtros de segurança. Neste cenário, o acesso diretamente à rede, sem roteamento pelo sistema operacional do domínio de controle, fica disponível como uma opção do administrador da Nuvem. Esta configuração pode ser realizada de duas formas:

¹⁷As topologias de rede disponíveis no Xen podem ser revisitadas na Subseção 2.2.3.

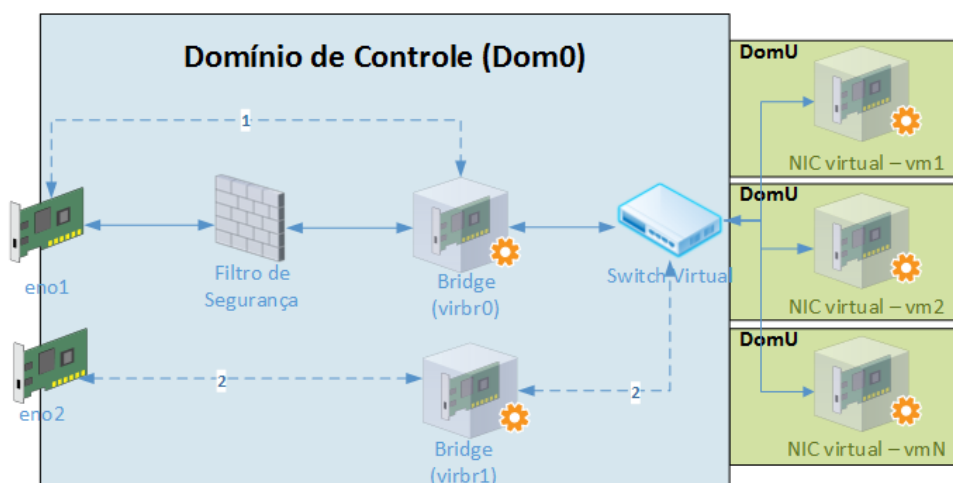


Figura 4.23: Topologia de rede da Nuvem de Simulação.

Fonte: Autor.

- a) Adicionando a interface de acesso à rede na configuração da ponte. Isto irá permitir que todos os domínios associados a esta ponte possam acessar diretamente a rede. Esta configuração está representada na Figura 4.23 pela seta tracejada número um.
- b) Criando uma nova ponte que inclui a interface da rede à qual deseja conceder o acesso e vinculando a(s) interface(s) do(s) domínio(s) a esta ponte. Esta configuração está representada na Figura 4.23 pela(s) seta(s) tracejada(s) número dois.

Para o acesso dos usuários aos domínios é realizado um redirecionamento de porta no domínio de controle para a porta 22 (SSH) do referido domínio. O acesso dos domínios à Internet é realizado via mascaramento de rede, também efetuado no domínio de controle.

4.5 CONDISERAÇÕES FINAIS

Neste capítulo, foi apresentada a proposta para uma Nuvem de Simulação Robótica capaz de fornecer ambientes para a simulação de VANTs. A plataforma de nuvem proposta foi apresentada considerando seus modelos arquitetural e de implementação. A implementação baseada no modelo proposto utilizou software de código-fonte aberto e gratuito, com o objetivo de atender às necessidades do Projeto DaaS do LaSiD UFBA. Todos os componentes dos modelos propostos, bem como da implementação realizada foram discutidos como parte deste trabalho.

CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho apresentou como proposta de plataforma robótica em nuvem destinada a oferta de ambientes para simulação baseada em VANTs.

A plataforma proposta foi desenvolvida utilizando software livre. O desenvolvimento de tal plataforma exigiu uma pesquisa exploratória prévia acerca das nuvens robóticas e tecnologias e conceitos associados, como a virtualização, computação em nuvem, sistemas de tempo real, ferramentas para programação e simulação robótica, dentre outros. Desta pesquisa foram selecionados os fundamentos teóricos mais relevantes para o entendimento da arquitetura proposta e implementada, sendo apresentados no [Capítulo 2](#).

No [Capítulo 3](#) foi apresentado o modelo de tarefas do UAV Parrot AR.Drone, que foi o resultado de uma extensa exploração deste modelo. Esta pesquisa associou o estudo de material teórico diverso, incluindo artigos, relatos da comunidade hacker interessada neste equipamento e registros de patentes com a observação experimental deste drone nas dependências do LaSiD e o estudo de códigos fonte criados por outras pesquisas ([PaparazziUAV, 2016](#); [Perquin, 2012](#)). Para facilitar pesquisas futuras, as informações obtidas sobre o funcionamento deste equipamento foram compiladas e disponibilizadas também nesta seção. A realização desta pesquisa contribuiu para maturar o entendimento sobre o problema trabalhado. Neste ponto, este trabalho também considerou alguns dos desafios impostos pelo uso do equipamento real citados anteriormente, o que também contribuiu para este entendimento.

Por fim, o [Capítulo 4](#) descreveu a proposta central deste estudo, uma Nuvem de Simulação Robótica, com arquitetura distribuída e, proposta para atender de forma rápida e flexível às demandas de simulação dos projetos de pesquisas envolvendo UAVs. Esta proposta teve como base o trabalho de [Agostinho et al. \(2011\)](#), sendo apoiado pelos demais trabalhos relacionados na [Seção 4.2](#). A plataforma proposta foi descrita em detalhes através de uma abordagem teórica, onde foi apresentada a arquitetura da Nuvem de Simulação e um modelo proposto para sua implementação, e uma abordagem prática, onde é feita a implementação utilizando software com código-fonte aberto e gratuito para atender demandas do Projeto DaaS, suportado pelo Laboratório de Sistemas Distribuídos (LaSiD) da UFBA.

Durante o desenvolvimento deste trabalho foram mapeados outros trabalhos que podem complementar o escopo deste estudo e que serão abordados em trabalhos futuros:

- Desenvolver os casos de uso ligados ao ator *Administrador* conforme o diagrama de casos de uso da [Figura 4.5](#) que é descrito na [Seção 4.4](#).
- Melhorar o desempenho do ambiente de simulação disponibilizado pela arquitetura proposta.
- Integrar a nuvem proposta neste trabalho com o RT-Openstack ([Xi et al., 2015](#)), uma extensão para o OpenStack que possibilita a distinção entre VMs com requisitos de tempo real e de propósito geral fazendo a correta distribuição destas entre os servidores de virtualização que podem lidar com este requisito. Desenvolvido pelos autores do RT-Xen, a integração com este é nativa.
- Construir e avaliar um ambiente de simulação com o *ROS 2.0*¹, ainda em desenvolvimento e com previsão de lançamento para o final de 2017. Esta versão tem em seu plano de desenvolvimento (roadmap) a adição de características interessantes para o modelo apresentado, tais como: programação de tarefas com requisitos de tempo real, execução na plataforma Windows, melhoria no mecanismo de comunicação adotando novo padrão e uma ponte de comunicação com o ROS 1.0, o que facilita a portabilidade de aplicações.
- Realizar uma pesquisa no código do simulador do AR.Drone (TUM Simulator) e modificá-lo para permitir que dois drones co-existam no mesmo cenário de simulação, superando o limite atual de apenas um drone por cenário.
- Implementar uma camada para simulação de rede entre os ambientes da nuvem para que sta possa também abrigar experimentos das pesquisas que envolvem os desafios de rede que o *Projeto DaaS* precisa enfrentar.
- Neste trabalho, o modelo de tarefas do AR.Drone foi levantado e descrito. Fazer a implementação das tarefas apresentadas neste modelo e observar o comportamento do ambiente durante sua execução irá complementar este estudo ao delimitar os tipos de tarefas que podem ser executadas sem perda de *deadline*. Ainda é salutar realizar uma comparação entre a implementação destas tarefas no ROS 1.0 e no ROS 2.0, utilizando seus resultados para definir os cenários onde cada versão melhor se aplica sob o aspecto de tempo real.

¹O ROS 2.0 está disponível em (<https://github.com/ros2/ros2/wiki>).

REFERÊNCIAS BIBLIOGRÁFICAS

- AGOSTINHO, L. et al. A cloud computing environment for supporting networked robotics applications. In: IEEE. *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*. [S.l.], 2011. p. 1110–1116.
- ARMBRUST, M. et al. *Above the Clouds: A Berkeley View of Cloud Computing*. [S.l.], 2009. Disponível em: <http://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html>.
- BARHAM, P. et al. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, ACM, New York, NY, USA, v. 37, n. 5, p. 164–177, out. 2003. ISSN 0163-5980. Disponível em: <http://doi.acm.org/10.1145/1165389.945462>.
- BELLARD, F. et al. *QEMU open source processor emulator*. 2017. 5 p. Disponível em: <URL:http://www.qemu.org>.
- BERGNER, B. S. F.; Regelungstechnik, S.-u. Quadrocopter stabilization using neuromorphic embedded dynamic vision sensors (eDVS). Agosto 2012. Disponível em: http://www.nst.ei.tum.de/fileadmin/w00bqs/www/pdf/papers_axenie/supervised_students/onboard_ardrone_edvs_stabilization.pdf.
- BRESCIANI, T. Modelling, identification and control of a quadrotor helicopter. *MSc Theses*, 2008.
- BRISTEAU, P.-J. et al. The navigation and control technology inside the ar. drone micro uav. *IFAC Proceedings Volumes*, Elsevier, v. 44, n. 1, p. 1477–1484, 2011.
- BUTTAZZO, G. *Hard real-time computing systems: predictable scheduling algorithms and applications*. [S.l.]: Springer Science & Business Media, 2011.
- CARDOZO, E. et al. A platform for networked robotics. In: IEEE. *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. [S.l.], 2010. p. 1000–1005.
- CASTRO, C. A. L.; ALMEIDA, W. R. M.; FILHO, P. M. de A. Computação em nuvem: Uma visão geral. *Acta Brazilian Science*, v. 1, n. 2, p. 62–69, 2013.
- CHAPERON, C.; PIERRE, E. *Method of synchronized control of electric motors of a remote-controlled rotary wing drone such as a quadricopter*. Google Patents, 2011. US Patent App. 13/118,367. Disponível em: <https://www.google.com/patents/US20110301787>.

- CHEN, Y.; DU, Z.; GARCÍA-ACOSTA, M. Robot as a service in cloud computing. In: IEEE. *Service Oriented System Engineering (SOSE), 2010 Fifth IEEE International Symposium on*. [S.l.], 2010. p. 151–158.
- CHIUEH, S. N. T.-c.; BROOK, S. A survey on virtualization technologies. *RPE Report*, p. 1–42, 2005.
- COOK, K. L. B. The silent force multiplier: The history and role of uavs in warfare. In: *2007 IEEE Aerospace Conference*. [S.l.: s.n.], 2007. p. 1–7. ISSN 1095-323X.
- DERBANNE, T. *Method of evaluating the horizontal speed of a drone, in particular a drone capable of performing hovering flight under autopilot*. Google Patents, 2013. US Patent 8,498,447. Disponível em: <https://www.google.com/patents/US8498447>.
- DILLON, T.; WU, C.; CHANG, E. Cloud computing: issues and challenges. In: IEEE. *2010 24th IEEE international conference on advanced information networking and applications*. [S.l.], 2010. p. 27–33.
- DU, Z. et al. Design of a robot cloud center. In: IEEE. *Autonomous Decentralized Systems (ISADS), 2011 10th International Symposium on*. [S.l.], 2011. p. 269–275.
- DUAN, Y. et al. Everything as a service (xaas) on the cloud: Origins, current and future trends. In: *2015 IEEE 8th International Conference on Cloud Computing*. [S.l.: s.n.], 2015. p. 621–628. ISSN 2159-6182.
- FARINES, J.-M.; FRAGA, J. d. S.; OLIVEIRA, R. d. Sistemas de tempo real. *Escola de Computação*, v. 2000, p. 201, 2000.
- Federal Communications Commission. *FCC ID RKXMYKONOS AR DRONE WITH WIFI MODULE by PARROT*. 2010. Disponível em: <https://fccid.io/RKXMYKONOS>.
- FIGUEIREDO, R.; DINDA, P. A.; FORTES, J. Guest editors' introduction: Resource virtualization renaissance. *Computer*, IEEE, v. 38, n. 5, p. 28–31, 2005.
- FOUNDATION, O. S. R. *ROS.org | Is ROS For Me?* 2017. Disponível em: <http://www.ros.org/is-ros-for-me/>.
- FREEBSD. *The FreeBSD Documentation Project - FreeBSD Handbook*. 2017. Disponível em: <https://www.freebsd.org/doc/handbook/jails.html>.
- FREEMAN, E.; FREEMAN, E. Use a cabeça–padrões de projeto. *Use a cabeça: Padrões de Projeto*, Alta Books Rio de Janeiro, 2007.
- GAMMA, E. *Design patterns: elements of reusable object-oriented software*. [S.l.]: Pearson Education India, 1995.
- GOLDBERG, K.; KEHOE, B. Cloud robotics and automation: A survey of related work. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2013-5*, 2013.

- GU, Z.; ZHAO, Q. A state-of-the-art survey on real-time issues in embedded systems virtualization. Scientific Research Publishing, 2012.
- GUIZZO, E. Robots with their heads in the clouds. *IEEE Spectrum*, IEEE, v. 48, n. 3, 2011.
- HOFFMANN, G. M. et al. Quadrotor helicopter flight dynamics and control: Theory and experiment. In: *Proc. of the AIAA Guidance, Navigation, and Control Conference*. [S.l.: s.n.], 2007. v. 2, p. 4.
- HSU, J.; KOENIG, N.; COLEMAN, D. *gazebo_ros_pkgs - ROS Wiki*. 2016. Disponível em: http://wiki.ros.org/gazebo_ros_pkgs.
- HU, G.; TAY, W. P.; WEN, Y. Cloud robotics: architecture, challenges and applications. *IEEE network*, IEEE, v. 26, n. 3, 2012.
- HUANG, H.; STURM, J. *tum_simulator - ROS Wiki*. 2014. Disponível em: http://wiki.ros.org/tum_simulator.
- IEEE. *IEEE Society of Robotics and Automation's Technical Committee on: Networked Robots*. 2015. Disponível em: <http://www-users.cs.umn.edu/~isler/tc/>.
- IETF. *Hypertext Transfer Protocol – HTTP/1.1*. 1999. Disponível em: <https://tools.ietf.org/html/rfc2616>.
- ISO. *ISO 8373:2012, Robots and robotic devices – Vocabulary*. 2012. Disponível em: <http://www.iso.org>.
- KAMEI, K. et al. Cloud networked robotics. *IEEE Network*, IEEE, v. 26, n. 3, 2012.
- KERSTAN, T. *Towards full virtualization of embedded real-time systems*. Tese, 2011.
- KOENIG, N.; HOWARD, A. Design and use paradigms for gazebo, an open-source multi-robot simulator. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. [S.l.: s.n.], 2004. v. 3, p. 2149–2154 vol.3.
- KOHORN, D. von. *GitHub - dougvk/tum_simulator: ROS indigo compatible tum_simulator*. 2015. Disponível em: https://github.com/dougvk/tum_simulator.
- KOUBÁA, A. A service-oriented architecture for virtualizing robots in robot-as-a-service clouds. In: SPRINGER. *International Conference on Architecture of Computing Systems*. [S.l.], 2014. p. 196–208.
- KRAJNÍK, T. et al. AR-Drone as a Platform for Robotic Research and Education. In: *Research and Education in Robotics: EUROBOT 2011*. Heidelberg: Springer, 2011.
- KUFFNER, J. J. Cloud-enabled robots. *IEEE-RAS International Conference on Humanoid Robots*, 2010.

LEE, M. et al. Supporting soft real-time tasks in the xen hypervisor. In: ACM. *ACM Sigplan Notices*. [S.l.], 2010. v. 45, n. 7, p. 97–108.

Linux Containers. *Linux Containers*. 2017. Disponível em: <https://linuxcontainers.org/>.

LORENCIK, D.; SINCAK, P. Cloud robotics: Current trends and possible use as a service. In: IEEE. *Applied Machine Intelligence and Informatics (SAMi), 2013 IEEE 11th International Symposium on*. [S.l.], 2013. p. 85–88.

MACÊDO, R. et al. Tratando a previsibilidade em sistemas de tempo-real distribuídos: Especificação, linguagens, middleware e mecanismos básicos (minicurso). *XXII Simpósio Brasileiro de Redes de Computadores*. Gramado, RS, Brasil, 2004.

MELL, P. M.; GRANCE, T. *SP 800-145. The NIST Definition of Cloud Computing*. Gaithersburg, MD, United States, 2011.

MENASCÉ, D. A. Virtualization: Concepts, applications, and performance modeling. In: *Int. CMG Conference*. [S.l.: s.n.], 2005. p. 407–414.

MEYER, J. *hector_quadrotor_gazebo_plugins - ROS Wiki*. 2013. Disponível em: http://wiki.ros.org/hector_quadrotor_gazebo_plugins.

MEYER, J.; KOHLBRECHER, S. *hector_quadrotor - ROS Wiki*. 2014. Disponível em: http://wiki.ros.org/hector_quadrotor.

MEYER, J. et al. Comprehensive simulation of quadrotor uavs using ros and gazebo. In: *3rd Int. Conf. on Simulation, Modeling and Programming for Autonomous Robots (SIMPAN)*. [S.l.: s.n.], 2012. p. to appear.

MONAJJEMI, M. et al. *ardrone autonomy: A ros driver for ardrone 1.0 & 2.0*. 2012. Disponível em: <http://ardrone-autonomy.readthedocs.io/en/latest/>.

Oracle Corporation. *The InnoDB Storage Engine*. 2017. Disponível em: <https://dev.mysql.com/doc/refman/5.7/en/innodb-storage-engine.html>.

Oracle Corporation. *MySQL*. 2017. Disponível em: <http://www.mysql.com/>.

OSRF. *Gazebo : Tutorials*. 2014. Disponível em: http://gazebosim.org/tutorials?cat=connect_ros.

OSRF. *Gazebo : Tutorials : Overview*. 2014. Disponível em: http://gazebosim.org/tutorials?cat=guided_b&tut=guided_b1.

PaparazziUAV. 2016. Disponível em: http://wiki.paparazziuav.org/wiki/Main_Page.

Parrot Inc. *CES 2010: Parrot AR.Drone allows video games to become reality*. 2010. Disponível em: <http://blog.parrot.com/2010/01/06/ces-2010-parrot-ardrone-allows-video-games-to-become-reality/>.

- PERQUIN, H. *AR.Drone Parrot » Tech Toy Hacks*. 2012. Disponível em: <http://blog.perquin.com/blog/category/ardrone/>.
- PISKORSKI, S. et al. *AR.Drone Developer Guide*. [S.l.], 2012.
- PLEBAN, J.-S.; BAND, R.; CREUTZBURG, R. Hacking and securing the ar. drone 2.0 quadcopter: investigations for improving the security of a toy. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. *IS&T/SPIE Electronic Imaging*. [S.l.], 2014. p. 90300L–90300L.
- POPEK, G. J.; GOLDBERG, R. P. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, ACM, v. 17, n. 7, p. 412–421, 1974.
- QUIGLEY, M. et al. Ros: an open-source robot operating system. In: KOBE. *ICRA workshop on open source software*. [S.l.], 2009. v. 3, n. 3.2, p. 5.
- RoboEarth Project. *RoboEarth | A World Wide Web for Robots*. 2014. Disponível em: <http://roboearth.ethz.ch/motivation>.
- ROSENBLUM, M.; GARFINKEL, T. Virtual machine monitors: Current technology and future trends. *Computer*, IEEE, v. 38, n. 5, p. 39–47, 2005.
- SANDHU, R.; FERRAILOLO, D.; KUHN, R. The nist model for role-based access control: towards a unified standard. In: *ACM workshop on Role-based access control*. [S.l.: s.n.], 2000. v. 2000, p. 1–11.
- SMITH, J. E.; NAIR, R. The architecture of virtual machines. *Computer*, IEEE, v. 38, n. 5, p. 32–38, 2005.
- TANENBAUM, A. S.; FILHO, N. M. *Sistemas operacionais modernos*. [S.l.]: Prentice-Hall, 1995.
- TAURION, C. Cloud computing: computação em nuvem: transformando o mundo da tecnologia da informação. *Rio de Janeiro: Brasport*, v. 2, n. 2, p. 2–2, 2009.
- The Linux Foundation. *What Is Open vSwitch? — Open vSwitch 2.8.90 documentation*. 2016. Disponível em: <http://docs.openvswitch.org/en/latest/intro/what-is-ovs/>.
- The Linux Foundation. *Home - The Linux Foundation*. 2017. Disponível em: <http://www.linuxfoundation.org>.
- The Open Group. *The X Window System*. 2017. Disponível em: <http://www.opengroup.org/desktop/x/>.
- VAQUERO, L. M. et al. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, ACM, v. 39, n. 1, p. 50–55, 2008.
- VLASENKO, D. *BusyBox: The Swiss Army Knife of Embedded Linux*. 2017. Disponível em: <https://www.busybox.net/about.html>.

VMWare Inc. *Server Virtualization Software — vSphere — VMware*. 2017. Disponível em: [⟨https://www.vmware.com/products/vsphere.html⟩](https://www.vmware.com/products/vsphere.html).

W3C. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. 2008. Disponível em: [⟨https://www.w3.org/TR/xml/⟩](https://www.w3.org/TR/xml/).

WINEHQ. *WineHQ - About Wine*. 2017. Disponível em: [⟨https://www.winehq.org/about⟩](https://www.winehq.org/about).

Xen Project Community. *Dom0 - Xen*. 2015. Disponível em: [⟨https://wiki.xen.org/wiki/Dom0⟩](https://wiki.xen.org/wiki/Dom0).

Xen Project Community. *Paravirtualization_(PV) - Xen*. 2015. Disponível em: [⟨https://wiki.xenproject.org/wiki/Paravirtualization_\(PV\)⟩](https://wiki.xenproject.org/wiki/Paravirtualization_(PV)).

Xen Project Community. *Xen Project Software Overview - Xen*. 2016. Disponível em: [⟨https://wiki.xenproject.org/wiki/Xen_Project_Software_Overview⟩](https://wiki.xenproject.org/wiki/Xen_Project_Software_Overview).

Xen Project Community. *Credit Scheduler - Xen*. 2017. Disponível em: [⟨https://wiki.xen.org/wiki/Credit_Scheduler⟩](https://wiki.xen.org/wiki/Credit_Scheduler).

Xen Project Community. *Xen Networking*. 2017. Disponível em: [⟨https://wiki.xenproject.org/wiki/Xen_Networking⟩](https://wiki.xenproject.org/wiki/Xen_Networking).

XI, S. et al. Rt-open stack: Cpu resource management for real-time cloud computing. In: IEEE. *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*. [S.l.], 2015. p. 179–186.

XI, S. et al. Rt-xen: Towards real-time hypervisor scheduling in xen. In: IEEE. *Embedded Software (EMSOFT), 2011 Proceedings of the International Conference on*. [S.l.], 2011. p. 39–48.

XI, S. et al. Real-time multi-core virtual machine scheduling in xen. In: IEEE. *Embedded Software (EMSOFT), 2014 International Conference on*. [S.l.], 2014. p. 1–10.

Yaskawa America, Inc. *Glossary of Robotics Terms*. 2017. Disponível em: [⟨https://www.motoman.com/glossary⟩](https://www.motoman.com/glossary).

YATES, D. R.; VAESSEN, C.; ROUPRET, M. From leonardo to da vinci: the history of robot-assisted surgery in urology. *BJU international*, Wiley Online Library, v. 108, n. 11, p. 1708–1713, 2011.

ZHANG, Q.; CHENG, L.; BOUTABA, R. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, Springer, v. 1, n. 1, p. 7–18, 2010.

A.1 DIAGRAMAS DE SEQUÊNCIA

- a) Carregar Parâmetros do Sistema (Figura A.1): O diagrama de sequência deste caso de uso engloba as interações entre quatro classes, sendo que duas (*SystemParameters* e *FileContent*) são classes para manipulação de dados. Como os parâmetros são persistidos em um arquivo texto, a classe *OSOperation*, responsável por executar operações ligadas ao Sistema Operacional, é observado um maior número de interações desta classe com as demais existentes neste diagrama. A classe *RTCloud* recebe as solicitações do ator Sistema e orquestra a execução da operação solicitada, devolvendo ao ator o resultado desta execução.
- b) Criar Novo Ambiente (Figura A.2): Este diagrama de sequência traz interações entre seis classes, dentre elas as já descritas *RTCloud* e *OSOperation*. Além da manipulação de arquivos esta última é utilizada neste caso de uso para a execução remota de comandos via protocolo SSH. Quanto a *RTCloud*, diferente do caso anterior onde o ator solicitava diretamente à classe *RTCloud* a execução de uma operação, esta solicitação é realizada primeiramente para a classe API, que desempenha a função da API de Gerenciamento descrita na Subseção 4.3.2. Sendo assim, a interação com a *RTCloud* é feita pela classe API. Nota-se que a classe *RoboticEnvironment* concentra o maior volume de interações com outras classes, pois contém o fluxo principal de execução deste caso de uso. A classe *EnvironmentDAO* tem como papel a interação com a Camada de Persistência de Dados, sendo utilizada para popular os atributos e objetos utilizados pela classe *RoboticEnvironment*. A classe *VirtualMachine* interage com a *OSOperation* para a execução de operações com máquinas virtuais como a criação, preparação e inicialização, dentre outros, através da execução remota de comandos.
- c) Remover Ambiente (Figura A.3): Este Caso de Uso utiliza os mesmos componentes do caso anterior. É possível verificar que existe uma maior atividade na classe *EnvironmentDAO* em relação ao Caso de Uso anterior devido à necessidade de realizar mais alterações nos registros existentes na camada de persistência (exclusão de VMs e de ambientes do banco de dados). Com isto, esta classe passa a se aproximar da classe *RoboticEnvironment* em termos de importância, porém esta também é neste Caso de Uso a classe principal.
- d) Verificar Permissão de Usuário (Figura A.4): Este caso de uso engloba cinco componentes, dentre eles as classes API e *RTCloud*, já discutidas anteriormente e que desempenham o mesmo papel dos casos anteriores. A classe *Security* é responsável

pela execução do fluxo principal de atividades deste Caso de Uso. A classe *SecurityDAO* é utilizada por esta para interação com a Camada de Persistência de Dados e instancia objetos da classe *SecurityRole* que serve tanto para armazenar dados referentes aos papel associado ao usuário quanto para verificação de regras de acesso (ACL).

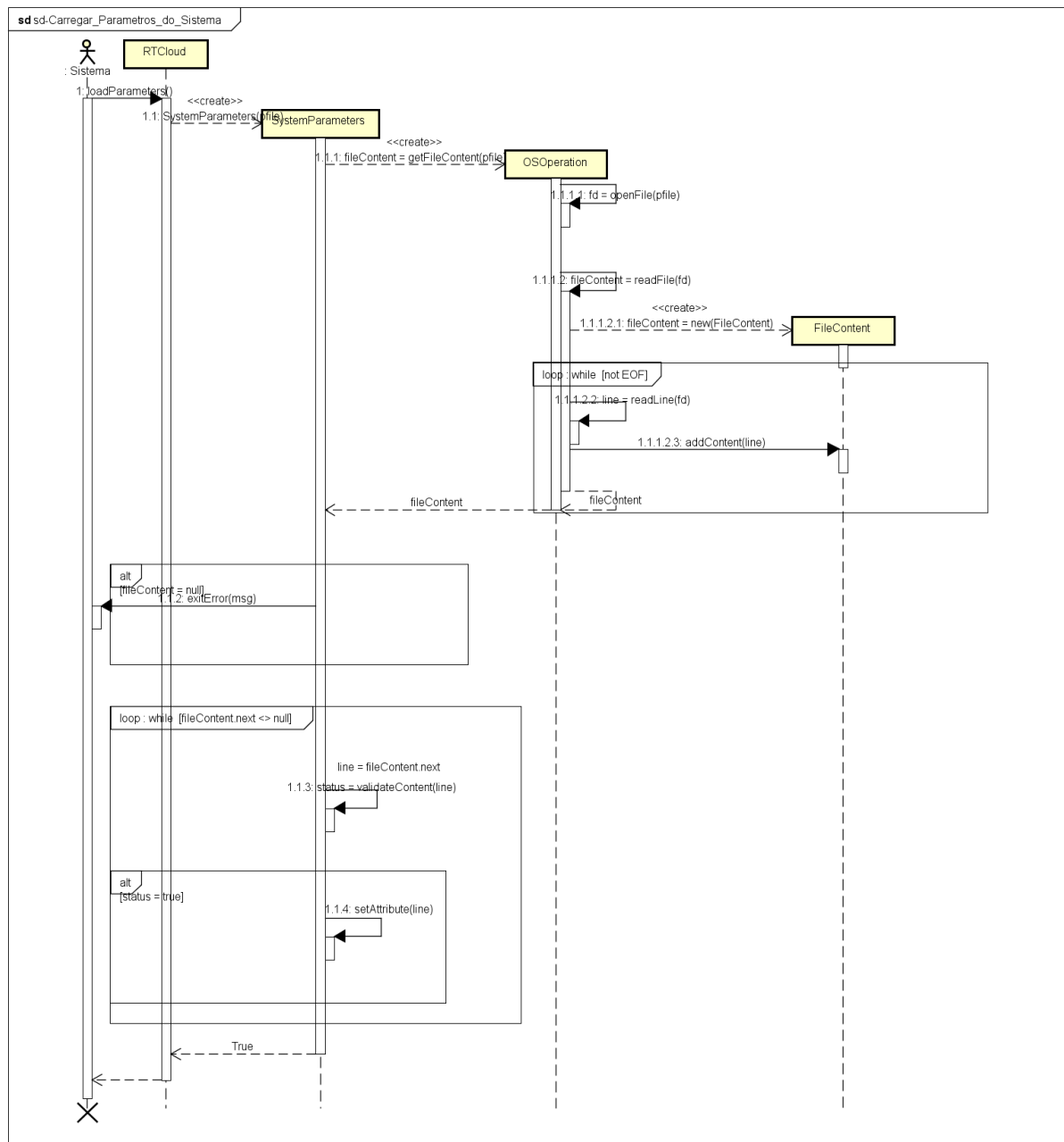


Figura A.1: Caso de uso: Carregar parâmetros do sistema.

Fonte: Autor.

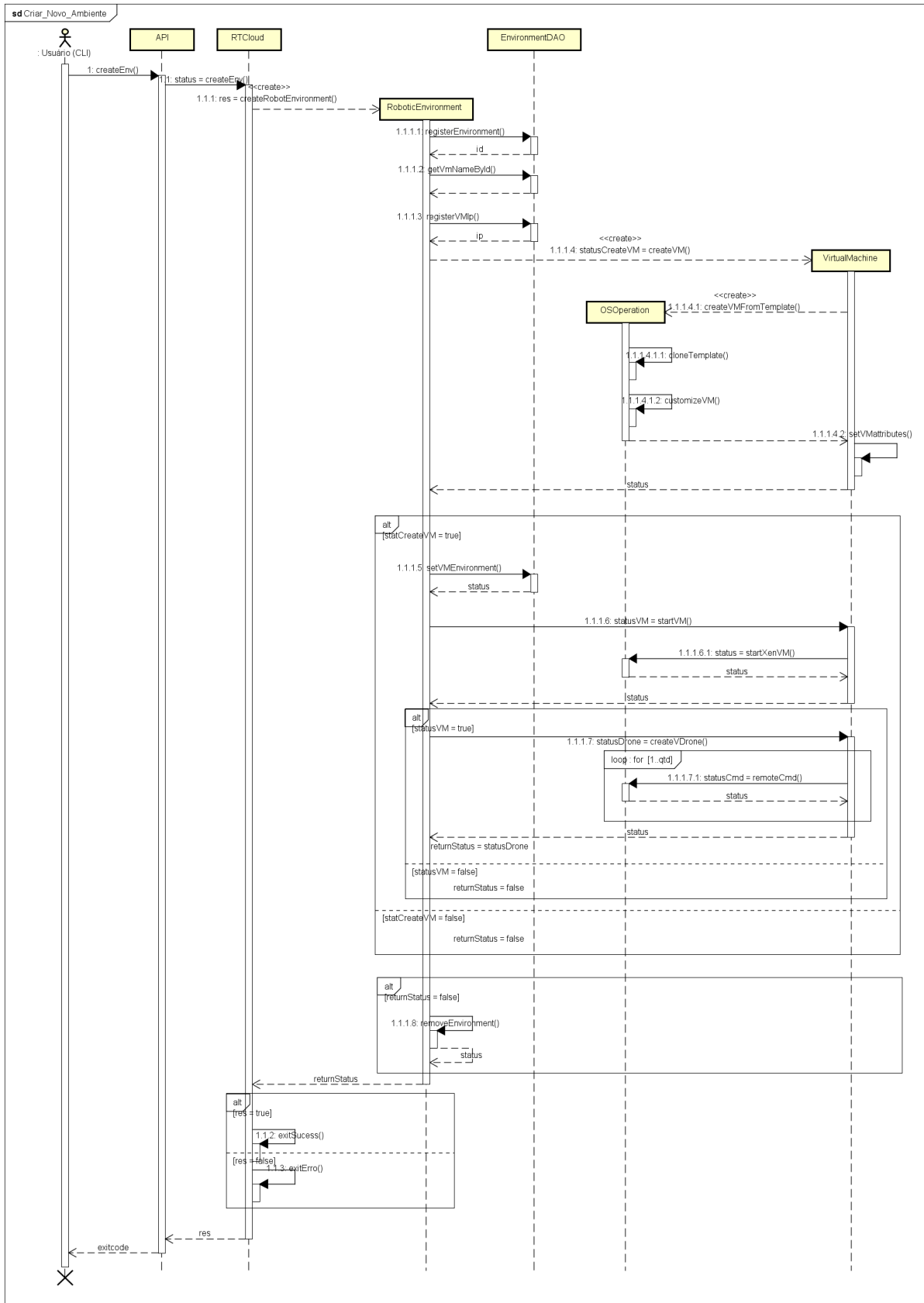


Figura A.2: Caso de uso: Criar novo ambiente.

Fonte: Autor.

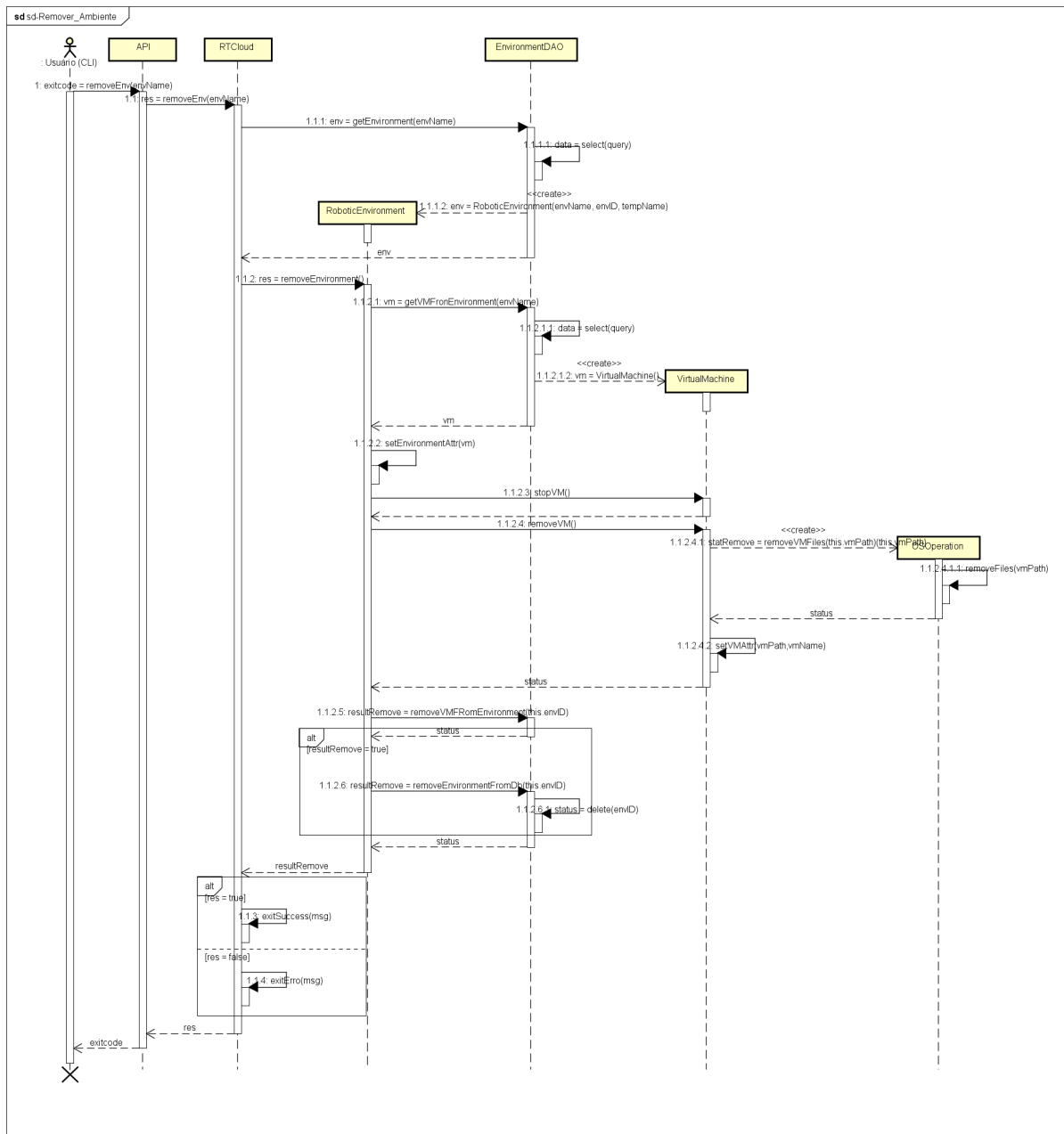


Figura A.3: Caso de uso: Remover Ambiente.

Fonte: Autor.

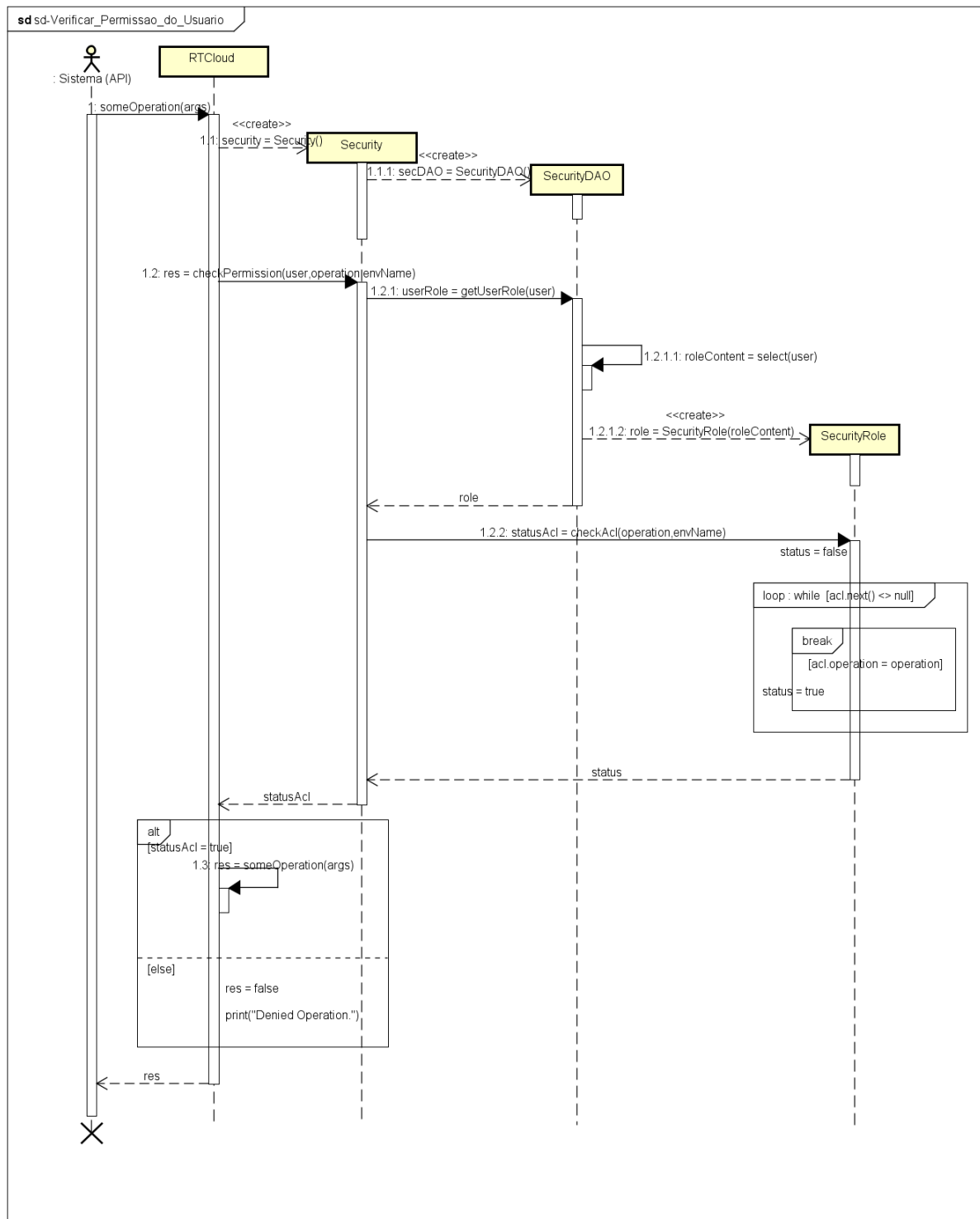


Figura A.4: Caso de uso: Verificar permissão do usuário.
 Fonte: Autor.

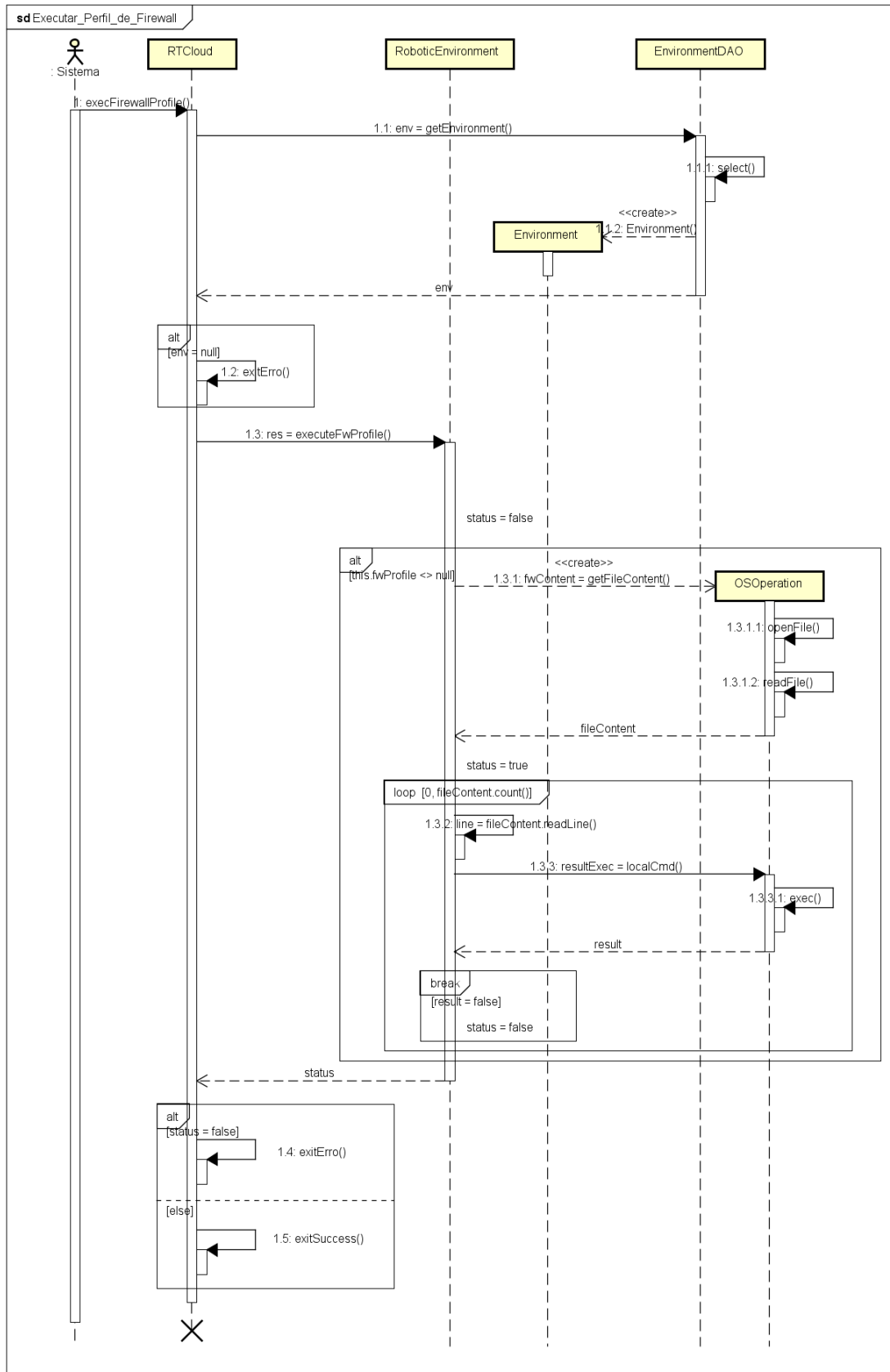


Figura A.5: Caso de uso: Executar perfil de firewall.
 Fonte: Autor.

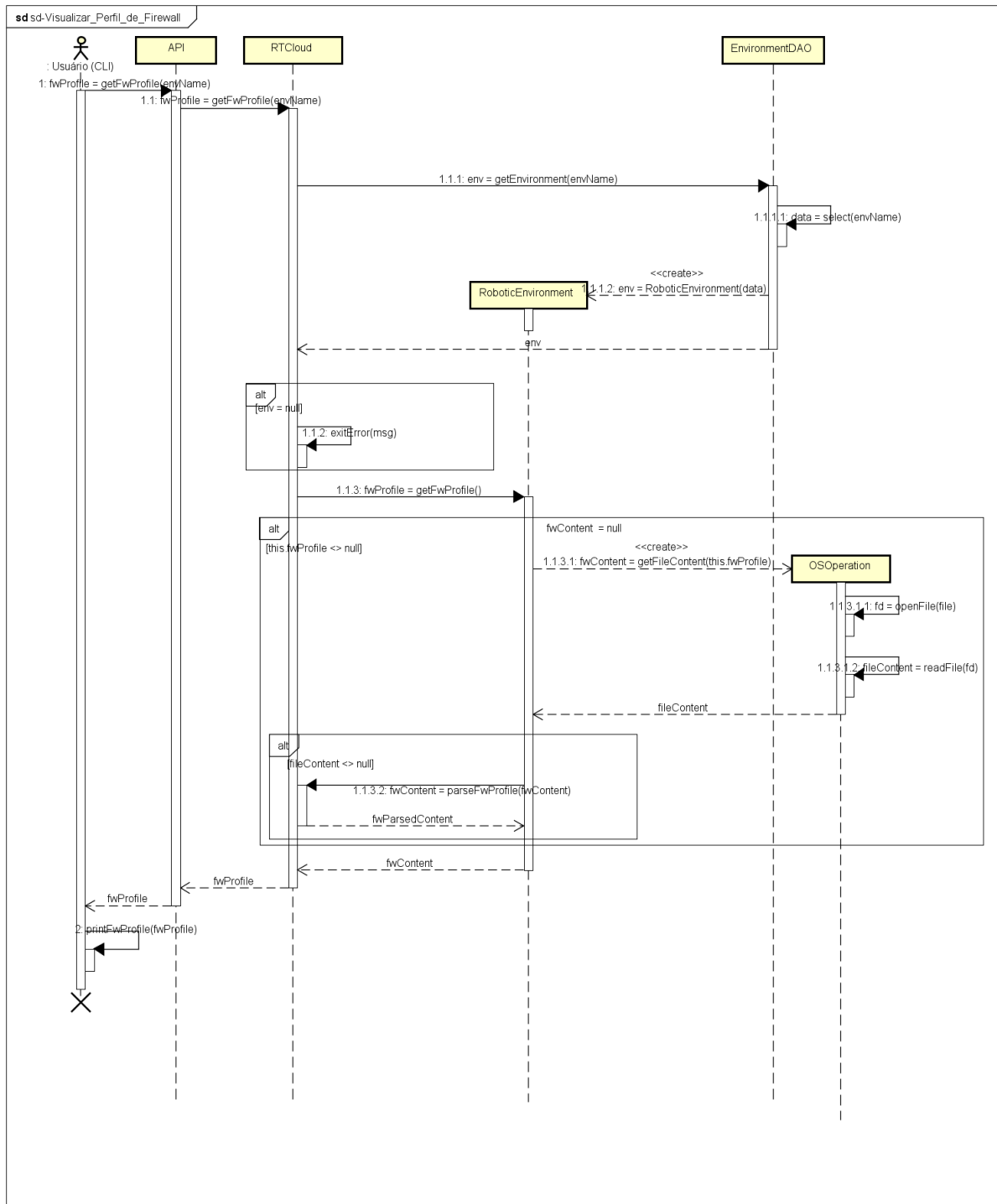


Figura A.6: Caso de uso: Visualizar perfil de firewall
 Fonte: Autor.

A.2 DIAGRAMA DE CLASSES

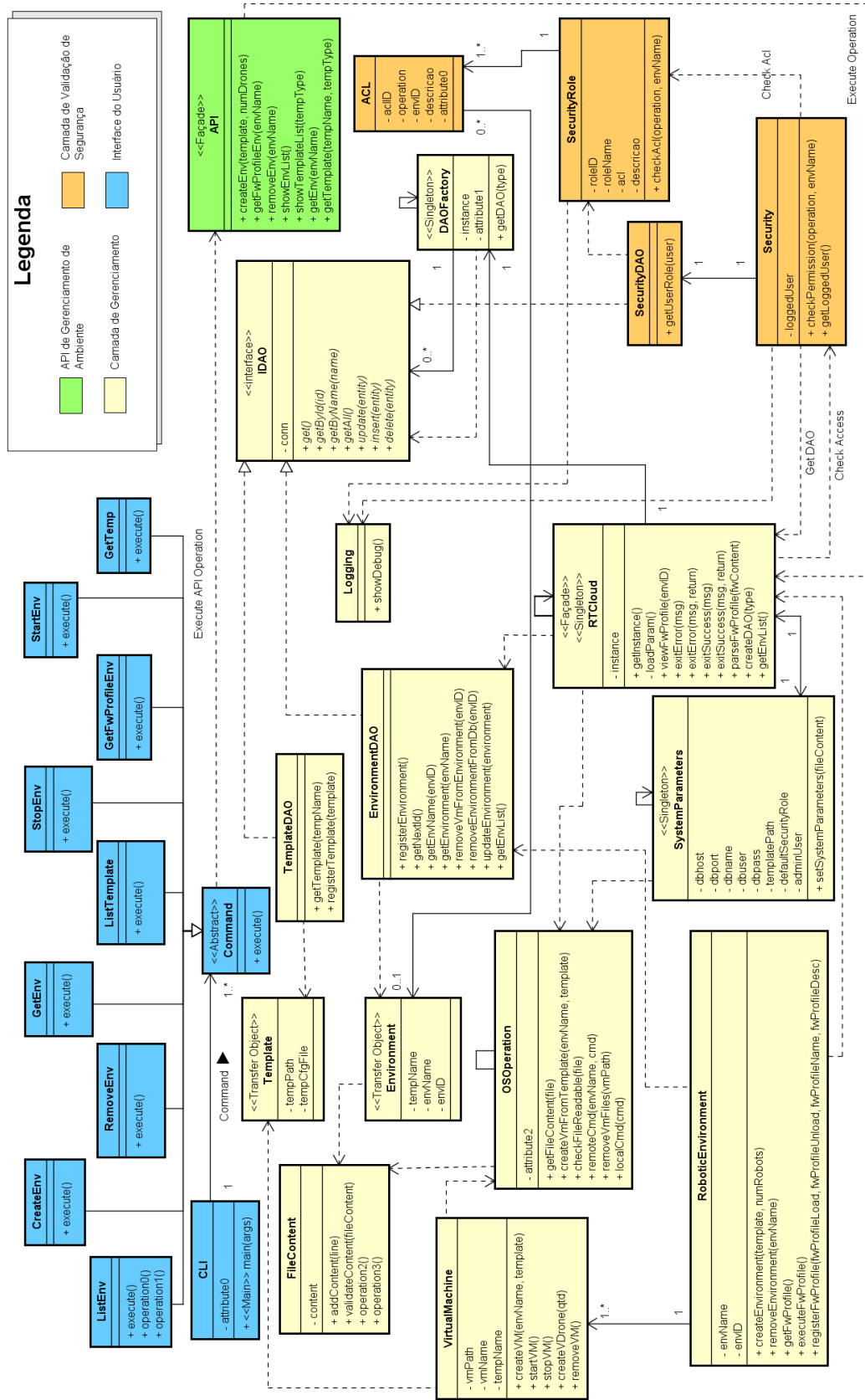


Figura A.7: Diagrama de Classes do *rtcloud-cli*.
Fonte: Autor.